Vladimir Melnic Ștefan Suceveanu

Aplicații cu microcontrolere de uz general

Editura AIT Laboratories
2003

Vladimir Melnic

Ştefan Suceveanu

Aplicaţii cu microcontrolere de uz general

CUPRINS

Introducere

Lucrarea de faţă se adresează unui cerc larg de cititori interesaţi de problemele ridicate de implementarea, în viaţa de zi cu zi, a unor dispozitive, aparate, sisteme care au în compunere elemente automate de coordonare, control, comandă etc. Se presupune că cititorii sunt familiarizaţi cu termenii specifici utilizatorilor de microprocesoare, pentru a câştiga în concizie unele expresii nefiind explicate.

Complexitatea sistemului poate fi extrem de variată, de exemplu, de la un filtru de cafea sau o maşină de spălat, până la un telefon mobil sau un automobil. Marele avantaj al microcontrolerelor faţă de microcalculatoarele clasice, cu microprocesor, constă în faptul că sistemul este astfel proiectat pentru a fi *prietenos*, aparatul *inteligent* fiind uşor de manevrat şi de către nespecialişti.

Spre deosebire de calculatoare (sau *computer* – în limba engleză desemnând calcule, lucrul cu numere), controlerul mai degrabă lucrează cu informaţii despre sistemul controlat: care este starea unui motor electric, temperatura unui lichid etc., funcţie de acestea şi de algoritmul de lucru programat luând deciziile necesare. Deci, controlerele pot fi considerate calculatoare obişnuite în care predomină interfeţele către exterior.

Istoria timpurie a calculatoarelor poate fi considerată începând cu Charles Babbage, un inventator britanic, autorul *maşinii analitice* în anul 1830. Ideea teoretică descria principii asemănătoare cu ceea ce fac şi calculatoarele din ziua de astăzi dar, dezvoltarea tehnologică din secolul trecut nu a permis realizarea practică a maşinii.

O idee mai practică a avut-o americanul Hermann Hollerith care a patentat o maşină de calculat în anul 1889. Maşina lui Hollerith lucra cu cartele perforate şi a fost utilizată ani de zile în scopuri statistice. Compania lui Hollerith, denumită *Tabulating Machine Company* a fost absorbită în anul 1924 de altă firmă, formând împreună vestita *International Business Machines Corporation*.

Un pas esenţial în progresul calculatoarelor l-a constituit introducerea algebrei booleene, o algebră care lucrează numai cu două cifre 0 şi 1. Algebra dezvoltată de Boole a permis implementarea ulterioară a calculatoarelor electronice, sisteme uriaşe care conţineau milioane de mici comutatoare care nu puteau avea decât două stări: deschis (asociat de regulă cifrei 0) şi închis (1). Comutatoarele sunt cunoscute astăzi sub denumirea de biţi. În calculatoarele moderne, grupul de 8 biţi este un bloc fundamental şi are numele de octet (în limba engleză *byte* şi prescurtat B).

În Marea Britanie Alan Turing a pus bazele logicii simbolice în anul 1937 (articolul *On Computable Number*), printr-o analiză comparată cu activitatea mentală umană. De asemenea, el a fost primul care a introdus conceptul de

algoritm, o metodă de prelucrare a datelor stabilită de un operator uman, precum şi automatul capabil să execute instrucţiunile algoritmului.

Al doilea război mondial a constituit un stimul puternic pentru dezvoltarea tehnicii de calcul. În Statele Unite, Mark Aiken a realizat MARK 1, un calculator format din 3304 comutatoare. Destinaţia sa era, din păcate, strict militară: tabele de tragere pentru artilerie. Dezvoltarea teoretică adusă de Turing, a permis unei echipe britanice realizarea unei maşini automate bazate pe comutatoare cu tuburi electronice, maşină folosită pentru descifrarea mesajelor Enigma ale marinei militare germane. Din păcate, realizarea practică a unei maşini abstracte universale Turing a eşuat în anul 1946

Din punct de vedere istoric, primul calculator electronic universal din lume poate fi considerat ENIAC. Acesta a fost realizat de Statele Unite, conţinea 17 468 tuburi electronice ocupând o suprafaţă de 450 m^2. Din punct de vedere al puterii de calcul al calculatoarelor actuale, ENIAC era mai puţin performant decât un calculator de buzunar.

Un salt tehnologic extraordinar a fost realizat de Bell Laboratories prin invenţia tranzistorului, dispozitivul minune: mai rapid, mai mic şi mai ieftin decât tubul electronic. Progresele microelectronicii au permis realizarea, în anul 1958, de către firma Texas Instrument, a primului circuit integrat, o reuniune de tranzistoare, diode, rezistenţe şi condensatoare. Avântul tehnologic al microelectronicii, a făcut posibilă realizarea primelor calculatoare universale iniţial cu tranzistoare (generaţia a II-a), ulterior cu circuite integrate (generaţia a III-a).

În anul 1969 firma Intel a primit o comandă de circuite integrate pentru o structură necesară unui calculator. Structura nu prea a avut succes dar, ideea de a crea un calculator cu programul modificabil prin intermediul unei memorii externe, a permis ca în doi ani firma Intel să scoată pe piaţă primul microprocesor, Intel 4004. Acest prim procesor avea o memorie volatilă (RAM) de 32 biţi, o memorie nevolatilă pentru programe (ROM) de 1024 biţi, un registru de deplasare de 10 biţi, având un set de 16 instrucţiuni. Şi următorul contract al firmei Intel necesar pentru realizarea unor terminale finalizat cu microprocesorul Intel 8008 nu a avut, iniţial, succesul scontat. O politică de piaţă inteligentă, realizată prin vânzarea de kituri compuse din microprocesor, memorii externe şi circuite de suport a avut însă un succes deosebit: volumul livrărilor a atins 330 milioane dolari în 1979.

Până la sfârşitul anului 1975, pe piaţă se găseau deja 40 de tipuri diferite de microprocesoare. Situaţia a dus la dezvoltarea unor circuite periferice şi de suport specifice pentru fiecare firmă, multe dintre ele fiind gândite chiar înainte de lansarea microprocesorului.

Integrarea în acelaşi integrat a circuitelor periferice a condus la realizarea microcalculatorului pe o singură structură – microcontrolerul, Primul microcontroler, Intel 8048 (1971) avea următoarea structură: unitate

centrală, memorii RAM şi ROM, circuite de intrare-ieşire. Nici acest circuit nu a avut un mare succes. Abia în anul 1981, o dată cu noul model IBM PC, microcontrolerul 8048 şi-a dovedit versatilitatea, fiind folosit pentru controlul tastaturii acestor tipuri de calculatoare.

Succesul deosebit al acestui microcontroler a condus la dezvoltarea continuă a acestor dispozitive: au început să fie integrate periferice pentru comanda unor dispozitive de afişare, convertoare analog/numerice (ADC) şi numeric/analogice (DAC), numărătoare etc.

Astfel, unul din cele mai utilizate microcontrolere de 8 biţi a fost Intel 8051, bazat pe o structură de 8048 la care se adăugau şi o interfaţă serială asincronă, două numărătoare de 16 biţi, având de asemenea, capabilităţi sporite pentru memorii.

Având exemplul firmei Intel, majoritatea celorlalţi producători au început să producă circuite asemănătoare, unele dintre ele fiind nevoite să cumpere licenţa de producţie. Astfel, în anul 1991 Philips a achiziţionat licenţa, deja anticului 8051, producând seria de microcontrolere 8xC552. Acestea aveau un nucleu 8051, la care se mai adăuga un convertor analog/numeric cu 16 intrări, un numărător suplimentar cu registre de captură şi comparare, un numărător pentru iniţializare (watchdog), două ieşiri de impulsuri modulate în durată, o interfaţă serială sincronă pentru standardul I^2C. De asemenea, 8xC552 conţinea o memorie RAM internă de 256 octeţi, 83C552 o memorie PROM de 8 kB iar 87C552 o memorie EPROM de 8 kB.

Succesul procesoarelor de 8 biţi şi nevoia de creştere a capacităţii de calcul, a vitezei de lucru sau a mărimii memoriei, a impus dezvoltarea, de către marele firme, a unor circuite, din ce în ce mai performante: în anul 1974 apare primul procesor de 16 biţi PACE (*Processing and Control Circuit*), urmat imediat de mai cunoscutele Intel 8086, Motorola 68000 sau Zilog Z8000. Utilizarea acestor procesoare în microcalculatoarele dezvoltate de Apple sau IBM au obligat la trecerea în faza următoare: microprocesoarele de 32 de biţi (familia Intel iAPX432 – I 80x86 sau Motorola M 680x0).

Limitările tehnologice au impus o limită în dezvoltarea unor alte procesoare, preferându-se o reorganizare a logicii procesului: în loc de a dezvolta procesoare complexe, cu mii de instrucţiuni s-a simplificat la maxim structura internă a circuitului, reducând drastic setul de instrucţiuni. Avantajul este evident: în locul unui procesor cu multe instrucţiuni (CISC - *Complex Instruction Set Computing*) la care se consumă mult timp pentru fiecare instrucţiune, un procesor cu puţine instrucţiuni (RISC – *Reduced Instruction Set Computing*) va executa extrem de rapid orice instrucţiune. Simplificarea structurii interne prin micşorarea decodificatorului de instrucţiuni, a permis integrarea unor module noi, cum ar fi procesoarele de virgulă mobilă, creşterea vitezei de execuţie şi, nu în ultimul rând, mărirea magistralelor interne sau externe până la 128 de biţi. Realizări deosebite în domeniul procesoarelor RISC constituie familiile SPARC (Sun Microsystems), PowerPC

(Motorola), MIPS (Silicon Graphics), Alpha (fost Digital Equipment Corporation, actualmente proprietatea Compaq) etc. Dominarea procesoarelor RISC este contrazisă de o singură excepţie CISC notabilă: familia Intel Pentium.

Procesoarele RISC au permis realizarea unui *mainframe* impresionant: calculatorul Cray 3D, realizat de firma Cray Research în anul 1991 are în compunere 256 procesoare RISC de 64 biţi, o frecvenţă a ceasului de 300 MHz şi un hard-disk de 430 TB (430 000 000 MB).

Controlerele au urmat îndeaproape evoluţia procesoarelor: concomitent cu circuitele pe 16 biţi au apărut controlere similare (de exemplu Intel 80186); filozofia RISC a fost implementată şi în universul controlerelor prin circuitele Siemens 80C16x sau Motorola 68332.

În încheiere, trebuie amintit de circuitele DSP (*Digital Signal Processor*) o sinteză a procesoarelor standard şi a controlerelor. Aceste circuite, special proiectate pentru prelucrarea semnalelor în timp real, beneficiază de o unitate de calcul în virgulă fixă şi flotantă extrem de performantă, au integrate circuitele principale de interfaţă (memorii, convertoare analog/numerice şi numeric/analogice ultrarapide, comunicaţii seriale sincrone şi asincrone, numărătoare etc.), fiind capabile, de exemplu, să efectueze calculul pentru o transformată Fourier rapidă în 1024 de puncte în mai puţin de 1 μs.

Astfel, ultima creaţie a firmei Advanced Micro Device, procesorul Athlon, are un nucleu DSP care execută instrucţiuni specifice pentru: modem, Dolby stereo, fişiere MP3 şi software ADSL.

1. Familia de microcontrolere 8xC552

Familia 8xC552 este reprezentată de o serie de circuite de 8 biţi de înaltă performanţă destinate utilizării în aplicaţii în timp real, cum ar fi automatizări industriale, controlul unor sisteme automate, aparate de măsură şi control etc.

Această familie constituie o perfecţionare a controlerului 8051, în sensul adăugării de noi periferice, creşterii vitezei de lucru sau a integrării de facilităţi suplimentare. Setul de instrucţiuni al 8xC552 este compatibil cu cel al lui 8051, în sensul că programele pentru 8051 pot rula şi pe 8xC552.

Funcţie de tipul memoriei ROM interne, cei trei membri ai familiei 8xC552 sunt:

- 87C552, cu memorie EPROM de 8 kB;
- 83C552, cu memorie PROM de 8 kB;
- 80C552, fără memorie ROM internă.

În lucrare se va folosi în continuare termenul generic 8xC552 pentru toţi membrii familiei, cu excepţia indicată circuitele având structura şi instrucţiunile identice.

O modificare a familiei 8xC552 faţă de modelul iniţial 8051 constă în adăugarea a două moduri cu consum redus de energie selectabile prin program:

- inactiv (`IDLE`), care lasă în funcţiune numai componentele existente şi în 8051 (numărătoarele 0 şi 1, memoria RAM, interfaţa serială asincronă);
- oprit (`POWER-DOWN`), care blochează oscilatorul circuitului, lăsând activă numai memoria RAM.

Descrierea funcţională a pinilor circuitului este prezentată în figura 1.1.

Controlerul 8xC552 conţine:

- unitate centrală de 8 biţi;
- 256 octeţi de memorie RAM, suprapuşi cu 128 octeţi de memorie alocaţi registrelor speciale;
- controler de întreruperi;
- şase porturi de intrări/ieşiri digitale;
- două timere/numărătoare de 16 biţi;
- un numărător de 16 biţi cu registre de captură şi comparaţie;
- un timer pentru deblocarea sistemului (watchdog);
- un convertor analog/numeric de 10 biţi cu 8 intrări;
- două ieşiri de impulsuri modulate în durată (utilizabile pentru convertoare numeric/analogice);
- două interfeţe seriale (una asincronă, compatibilă RS-232, cealaltă sincronă, compatibilă I^2C).

Structura internă a controlerului 8xC552 este prezentată în figuraFigura 1.2.

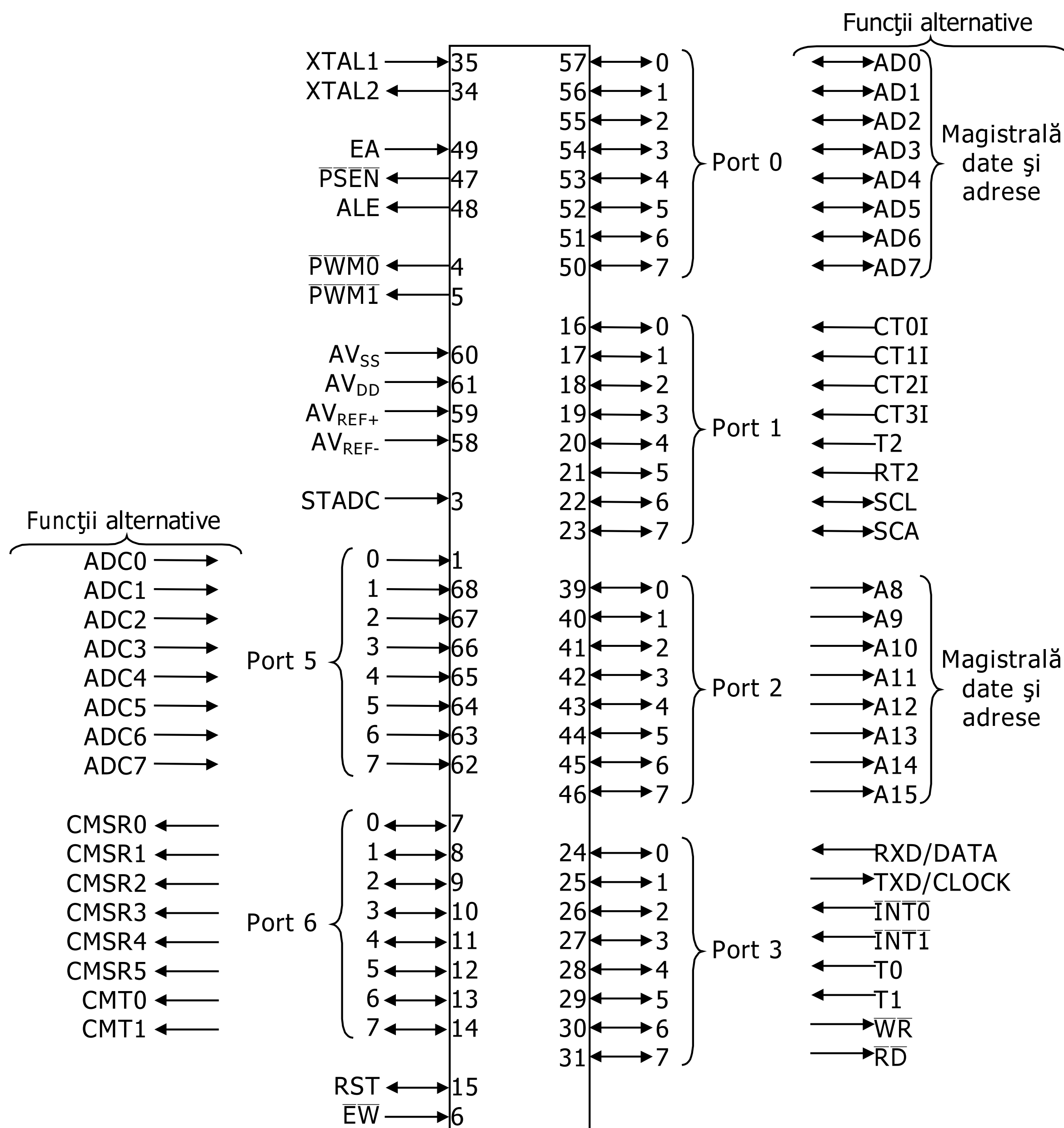


Figura 1.1. Microcontrolerul 8xC552

1.1. Unitatea centrală şi registrele sale

Funcţiile de bază ale unităţii centrale sunt: aducerea instrucţiunii din memoria program şi decodificarea ei, executarea operaţiilor aritmetice şi logice, memorarea rezultatelor anterioare, controlul perifericelor etc.

Sincronizarea funcţionării unităţii centrale şi a întregului sistem de periferice este asigurată de un oscilator stabilizat cu un cristal de cuarţ sau un rezonator ceramic. Controlerul poate lucra şi cu oscilator extern, situaţie în care semnalul este aplicat la pinul `XTAL1`. Pentru reducerea consumului de energie, un bit al registrului special `PCON` poate dezactiva oscilatorul circuitului, blocând funcţionarea tuturor modulelor (modul `POWER-DOWN`); totuşi, în acest mod este conservat conţinutul memoriei RAM.

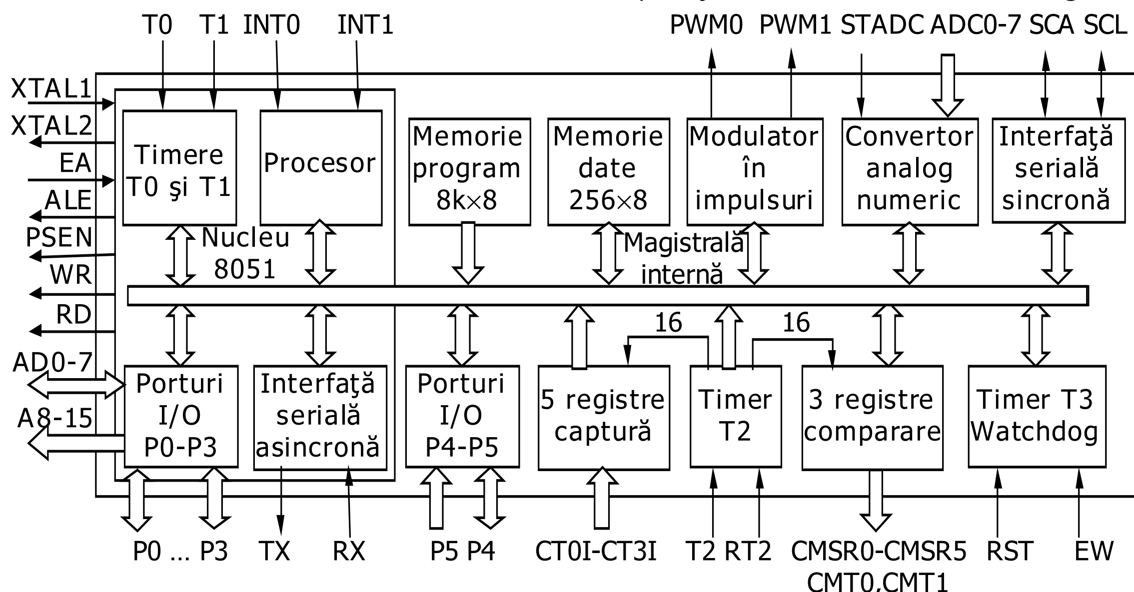


Figura 1.2. Structura internă a microcontrolerului 83C552

Un ciclu maşină al unităţii centrale este format din 12 oscilaţii complete, împărţite în 6 stări (S1-S6), fiecare stare fiind formată din două faze (P1 şi P2). Cele 12 oscilaţii sunt astfel codificate de la S1P1 la S6P2. Cele mai multe din instrucţiunile controlerului sunt executate într-un ciclu maşină (cu un oscilator de 12 MHz, un ciclu maşină reprezintă 1 μs), o altă parte sunt executate în două cicluri maşină (2 μs) iar înmulţirea şi împărţirea în 4μs.

Unitatea centrală are un contor program (PC) de 16 biţi, cu capacitate de adresare de 64 kB pentru memoria de program. Memoria de date, cu o capacitate maximă tot de 64 kB, este adresabilă în două moduri, prin intermediul unor registre interne: R0 sau R1, respectiv DPTR.

Unitatea centrală controlează următoarele blocuri de memorie:

- până la 64 kB memorie program (ROM);
- până la 64 kB memorie date (RAM);
- 256 octeţi memorie RAM internă, suprapuşi cu 128 octeţi pentru registrele speciale.

Selecţia modulului de memorie program, intern sau extern, este asigurată de semnalul EA: dacă acesta are valoarea 1 LOGIC este selectată memoria program internă (numai la 83C552); în cazul 0 LOGIC sunt aduse instrucţiuni numai din memoria externă, de la 0000h la 1FFFh. De menţionat faptul că zona de memorie program de la 0000h la 0002h este rezervată pentru adresa programului de iniţializare, în timp ce locaţiile de la 0003h la 0073h sunt folosite pentru adresele rutinelor de tratare a întreruperilor.

Diferenţierea tipului de memorie adresată (program sau date) este asigurată de starea semnalului $\overline{\text{PSEN}}$. Cu excepţia acestuia, în situaţia lucrului cu memorie program externă, când magistrala de date şi adrese este multiplexată pe P0 şi P2, separarea magistralei inferioare de adrese este asigurată de semnalul ALE.

Memoria de date interne este împărţită în trei secţiuni: un bloc de 128 octeţi la adrese mici, un bloc de 128 octeţi la adrese mari şi o zonă de 128 de octeţi destinată registrelor speciale.

Primul bloc de memorie (de la 00h la 07Fh) este adresabil direct sau indirect. Zona de memorie de la 00h la 1Fh conţine patru bancuri de registre, fiecare format din 8 registre, de la R0 la R7; la un moment dat, poate fi selectat un singur banc de registre prin intermediul a doi biţi definiţi în registrul special de stare (PSW). Următoarele 16 locaţii, de la 20h la 2Fh, conţin 128 de biţi adresabili direct. Această zonă este folosită, de regulă, pentru operaţii booleene sau pentru indicatoare de program. Zona de memorie de la 30h la 7Fh este utilizabilă pentru variabile.

Locaţiile de la 80h la FFh sunt împărţite între memoria de date şi registrele speciale iar adresarea lor poate fi indirectă (pentru memoria date) sau directă (pentru registrele speciale).

Controlerele din familia 8xC552 are posibilitatea să adreseze o memorie de date externe cu o capacitate de până la 64 kB. În acest scop sunt utilizate semnalele $\overline{PSEN}$ (selectare memorie date sau program), ALE (magistrala de adrese A0-A7 activă pe portul P0), $\overline{RD}$ şi $\overline{WR}$ (citire, respectiv scriere date).

Cu excepţia celor 256 de octeţi de memorie internă, unitatea centrală mai dispune de câteva registre importante:

a) acumulatorul A;
b) registrul B;
c) pointerul stivei SP;
d) registrul de stare PSW;
e) registrului pointer la memoria de date DPTR.

a) Acumulatorul

Acest registru este direct adresabil şi este referit mnemonic ca A. Este adresabil la nivel de bit. Este registrul principal de lucru al unităţii aritmetice şi logice (ALU). De asemenea, prin intermediul său se pot face schimburi de date cu memoria externă, salturi relative la conţinutul său etc.

b) Registrul B

Este un registru adresabil la nivel de bit. Este destinat ca registru auxiliar pentru operaţiile de înmulţire şi împărţire dar poate fi folosit şi ca registru general.

c) Pointerul stivei

Acest registru de 8 biţi este folosit pentru poziţionarea stivei în memoria internă. Mărimea stivei este de maxim 256 octeţi şi este limitată de memoria RAM internă disponibilă.

d) Registrul de stare

Acest registru de 8 biţi conţine informaţii referitoare la starea programului. Structura sa este prezentată în tabelul 1.1.

Tabelul 1.1

PSW (D0h)	CY	AC	F0	RS1	RS0	OV	F1	P

<table>
<tr><td>CY</td><td colspan="3">Indicator transport (carry).</td></tr>
<tr><td>AC</td><td colspan="3">Indicator transport auxiliar (pentru operaţiile BCD).</td></tr>
<tr><td>F0</td><td colspan="3">Indicator utilizator 0 (de uz general).</td></tr>
<tr><td rowspan="6">RS1
RS0</td><td colspan="3">Biţi selectare banc registre de lucru:</td></tr>
<tr><td>RS1</td><td>RS0</td><td>Banc registre selectat</td></tr>
<tr><td>0</td><td>0</td><td>Banc 0 (00h – 07h)</td></tr>
<tr><td>0</td><td>1</td><td>Banc 1 (08h – 0Fh)</td></tr>
<tr><td>1</td><td>0</td><td>Banc 2 (10h – 17h)</td></tr>
<tr><td>1</td><td>1</td><td>Banc 3 (18h – 1Fh)</td></tr>
<tr><td>OV</td><td colspan="3">Indicator depăşire.</td></tr>
<tr><td>F1</td><td colspan="3">Indicator utilizator (de uz general).</td></tr>
<tr><td>P</td><td colspan="3">Indicator paritate acumulator.</td></tr>
</table>

e) Registrul pointer la memoria de date

Registrul pointer la memoria de date (DPTR) este un registru de 16 biţi destinat lucrului cu memoria externă. Este format dintr-un octet inferior (DPL), respectiv unul superior (DPH).

DPTR permite accesul indirect la memoria de date externă sau la constante din memoria program. De asemenea, el poate fi utilizat ca un registru de 16 biţi sau ca două registre de 8 biţi independente.

Trebuie amintit că memoria de date externă poate transfera indirect un octet în acumulator nu numai folosind adresa locaţiei din DPTR (situaţie în care capacitatea de adresare este maximă – 64 kB) dar poate fi adresată şi pe 8 biţi, prin intermediul registrelor R0 sau R1 (situaţie în care capacitatea de adresare este de 256 octeţi). Mărirea spaţiului de adresare prin intermediul R0 sau R1 se poate face utilizând pentru adresarea memoriei externe biţi din porturile de intrare-ieşire neutilizaţi în alte scopuri.

1.2. Registrele speciale – SFR

Registrele speciale, poziţionate în memoria internă de date în domeniul 80h-FFh, conţin toate registrele sistemului, cu excepţia contorului program (PC) şi a celor 8 bancuri de registre. SFR sunt destinate controlului modulelor interne, stării controlerului, setării modurilor economice de lucru etc. Cele 56 de registre speciale ale familiei 8xC552 sunt prezentate în tabelul 1.2 clasificate după adresă. Pentru o identificare mai rapidă a registrelor, tabelul 1.3 le prezintă clasificate după funcţiuni.

Detaliile referitoare la seturile de registre speciale care controlează modulele interne ale familiei 8xC552 sunt prezentate în subcapitolele destinate fiecărui modul.

Tabelul 1.2

Simbol	Descriere	Adresă	Adresă bit, funcţie alternativă port							
T3	Timer 3	FFh								
PWMP	Prescaler PWM	FEh								
PWM1	Registru PWM 1	FDh								
PWM0	Registru PWM 0	FCh								
IP1	Registru priorităţi 1	F8h	PT2	PCM2	PCM1	PCM0	PCT3	PCT2	PCT1	PCT0
B	Registru B	F0h								
RTE	Comutare/validare comp. T2	EFh	TP47	TP46	RP45	RP44	RP43	RP42	RP41	RP40
STE	Setare registre comp. T2	EEh	TG47	TG46	SP45	SP44	SP43	SP42	SP41	SP40
TMH2	Registru T2	EDh								
TML2	Registru T2	ECh								
CTCON	Control captură T2	EBh	CTN3	CTP3	CTN2	CTP2	CTN1	CTP1	CTN0	CTP0
TM2CON	Registru control T2	EAh	T2IS1	T2IS0	T2ER	T2B0	T2P1	T2P0	T2MS1	T2MS0
IEN1	Validare întreruperi 1	E8h	ET2	ECM2	ECM1	ECM0	ECT3	ECT2	ECT1	ECT0
ACC	Acumulator	E0h								
S1ADR	Registru adresă slave I^{2}C	DBh	Adresă slave							GC
S1DAT	Registru date I^{2}C	DAh								
S1STA	Registru stare I^{2}C	D9h	SC4	SC3	SC2	SC1	SC0	0	0	0
S1CON	Registru control I^{2}C	D8h	–	EN51	STA	ST0	SI	AA	CR1	CR0
PSW	Registru stare program	D0h	CY	AC	F0	RS1	RS0	OV	F1	P
CTH3	Registru captură 3	CFh								
CTH2	Registru captură 2	CEh								
CTH1	Registru captură 1	CDh								
CTH0	Registru captură 0	CCh								
CMH2	Registru comparare 2	CBh								
CMH1	Registru comparare 1	CAh								
CMH0	Registru comparare 0	C9h								
TM2IR	Registru întreruperi T2	C8h	T2OV	CMI2	CMI1	CMI0	CTI3	CTI2	CTI1	CTI0
ADCH	Registru convertor A/D	C6h								
ADCON	Registru control ADC	C5h	ADC.1	ADC.0	ADEX	ADCI	ADCS	AADR2	AADR1	AADR0
P5	Port 5	C4h	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
P4	Port 4	C0h	CMT1	CMT0	CMSR5	CMSR4	CMSR3	CMSR2	CMSR1	CMSR0
IP0	Registru priorităţi 0	B8h	–	PAD	PS1	PS0	PT1	PX1	PT0	PX0
P3	Port 3	B0h	RD	WR	T1	T0	$\overline{INT1}$	$\overline{INT0}$	TXD	RXD
CTL3	Registru captură 3	AFh								
CTL2	Registru captură 2	AEh								
CTL1	Registru captură 1	ADh								
CTL0	Registru captură 0	ACh								
CML2	Registru comparare 2	ABh								
CML1	Registru comparare 1	AAh								
CML0	Registru comparare 0	A9h								
IEN0	Validare întreruperi 0	A8h	EA	EAD	ES1	ES0	ET1	EX1	ET0	EX0
P2	Port 2	A0h	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
S0BUF	Registru buffer UART	99h								
S0CON	Registru control UART	98h	SM0	SM1	SM2	REN	TB8	RB8	T1	R1
P1	Port 1	90h	SDA	SCL	RT2	T2	CT3I	CT2I	CT1I	CT01
TH1	Registru T1	8Dh								
TH0	Registru T0	8Ch								
TL1	Registru T1	8Bh								
TL0	Registru T0	8Ah								
TMOD	Registru mod T0 şi T1	89h	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TCON	Registru control T0 şi T1	88h	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
PCON	Registru control consum	87h	SMOD	–	–	WLE	GF1	GF0	PD	IDL
DPH	Registru pointer date	83h								
DPL	Registru pointer dat	82h								
SP	Indicator stivă	81h								
P0	Port 0	80h	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0

OBSERVAŢIE: *caracterele italice* indică registre “numai citire”.

Tabelul 1.3

<table>
<tr><td>Registre aritmetice
Acumulator, registru B, registru stare program (PSW)</td><td rowspan="2">Timere/Numărătoare
Timer T0 (TL0 şi TH0), timer T1 (TL1 şi TH1), registru mod T0 şi T1 (TMOD), registru control T0 şi T1 (TCON), timer T2 (TML2 şi TMH2), timer T3, registru control T2 (TM2CON)</td></tr>
<tr><td>Porturi
Port0 (P0), Port1 (P1), Port2 (P2), Port3 (P3), Port4 (P4), Port5 (P5)</td></tr>
<tr><td>Indicatori
Indicator stivă (SP), pointer date (DPH şi DPL)</td><td rowspan="3">Logică captură şi comparare
Control captură T2 (CTCON), registru întreruperi T2 (TM2IR), captură 0 (CTL0 şi CTH0), captură 1 (CTL1 şi CTH1), captură 2 (CTL2 şi CTH2), captură 3 (CTL3 şi CTH3), comparare 0 (CML0 şi CMH0), comparare 1 (CML1 şi CMH1), comparare 2 (CML2 şi CMH2), comutare/validare registre comparare T2 (RTE), setare registre comparare T2 (STE)</td></tr>
<tr><td>Modulatorul de impulsuri în durată
Prescaler PWM, registru PWM0, registru PWM1</td></tr>
<tr><td>Sistemul de întreruperi
Registru priorităţi 0 (IP0), registru priorităţi 1 (IP1), validare întreruperi 0 (IEN0), validare întreruperi 1 (IEN1)</td></tr>
<tr><td>Convertorul analog/numeric
Registru convertor A/D (ADCH), control ADC (ADCON)</td><td rowspan="2">Interfeţe seriale
Registru control UART (S0CON), buffer UART (S0BUF), registru control I^2C (S1CON), date I^2C (S1DAT), registru stare I^2C (S1STA), adresă slave I^2C (S1ADR)</td></tr>
<tr><td>Control consum energie
Registru control consum (PCON)</td></tr>
</table>

1.3. Setul de instrucţiuni

Setul de instrucţiuni al familiei 8xC552 este o modernizare a predecesorului său, 8051. Sunt introduse instrucţiuni suplimentare pentru controlul modulelor nou adăugate, precum şi câteva instrucţiuni noi: scădere cu împrumut, comparare, înmulţire şi împărţire.

Instrucţiunile controlerului acţionează la nivel de bit, 4 biţi, 8 biţi sau 16 biţi. Faţă de circuitul originar, 8051, modurile de adresare au fost diversificate, existând acum 5 tipuri:

a) adresare la registre;
b) adresare directă;
c) adresare indirectă;
d) adresare imediată;
e) adresare cu registru de bază şi index.

Adresarea registrelor permite accesul la cele opt registre R0-R7 selectate de doi biţi (RS0 şi RS1) din registrul PSW. Accesul la locaţiile respective de memorie (00h-1Fh) se poate face însă şi prin adresare directă, bancul de registre situându-se în zona 00h-7Fh care este adresabilă în mod direct. În zona registrelor speciale, registrele care controlează modulele interne sunt de asemenea adresabile direct.

În cadrul memoriei interne există două blocuri de 16 octeţi care sunt adresabile direct la nivel de bit: 128 de biţi pot fi adresaţi direct în zona (20h-2Fh), ceilalţi 128 de biţi găsindu-se la adresele 80h-FFh care sunt divizibile cu opt.

Adresarea indirectă este folosită pentru accesul zonei de memorie de la 80h la FFh care nu este adresată ca SFR, precum şi pentru memoria externă de date. Pentru memoria internă adresarea este făcută prin intermediul

registrelor R0 sau R1. Memoria externă este adresată fie prin registrele R0 şi R1 (caz în care blocul maxim de memorie este de 256 octeţi), fie prin registrul DPTR (caz în care capacitatea de adresare este de 64 kB).

Adresarea imediată este folosită pentru încărcarea de constante numerice ca parte a instrucţiunii din memoria program.

Adresarea cu registru de bază şi index este folosită pentru lucrul cu tabele de conversie, tabele de salturi etc. Pentru aceasta este folosit ca registru de bază DPTR sau PC, registrul index fiind acumulatorul.

Instrucţiunile familiei 8xC552 sunt optimizate atât din punct de vedere al lungimii codului (49 de instrucţiuni pe un octet, 45 pe doi octeţi şi 17 pe trei octeţi), cât şi al timpului de execuţie (64 de instrucţiuni sunt executate într-un ciclu maşină, 45 în două cicluri maşină şi două în 4 cicluri maşină).

Cele 111 instrucţiuni se pot clasifica în patru grupe funcţionale:

a) transfer de date;
b) instrucţiuni aritmetice;
c) instrucţiuni logice;
d) controlul programului.

a) Transferul de date

Operaţiunile de transfer de date sunt împărţite în trei clase:

- de uz general;
- specifice acumulatorului;
- adresare imediată pe 16 biţi.

Transferul de date de uz general constă în trei tipuri de operaţii:

MOV – realizează transferul unui bit sau octet de la operandul sursă la operandul destinaţie. Combinând operanzii şi modurile de adresare rezultă 57 de instrucţiuni diferite.

PUSH – incrementează registrul SP şi apoi transferă octetul desemnat de operandul sursă în memoria adresată de SP.

POP – transferă octetul desemnat în operandul sursă de la locaţia adresată de SP şi apoi decrementează registrul SP.

Operaţiile specifice acumulatorului sunt:

XCH – schimbă operandul sursă cu acumulatorul.

XCHD – schimbă 4 biţi ai operandul sursă cu 4 biţi ai acumulatorului.

MOVX – realizează transferul unui octet între memoria externă şi acumulator. Adresa externă este specificată de R0 şi R1 sau DPTR.

MOVC – realizează transferul unui octet între memoria program şi acumulator. Adresarea este cu registru de bază (PC sau DPTR) şi index (A).

Adresarea imediată pe 16 biţi semnifică încărcarea registrului DPTR cu o valoare pe doi octeţi.

b) Instrucţiuni aritmetice

Familia 8xC552 dispune de patru operaţii aritmetice de bază. Indicatorul depăşire (OV) poate fi folosit pentru operaţii de adunare şi scădere pentru numere în cod BCD cu semn sau fără semn; de asemenea, pentru numerele BCD există o instrucţiune pentru corecţia acumulatorului.

Cu excepţia indicatorului OV mai există încă trei indicatori care reflectă rezultatul operaţiei:

- C (transport) – setat dacă în urma operaţiei rezultă o depăşire a acumulatorului;
- AC (transport auxiliar) – setat dacă rezultă un transport între biţii 3 şi 4 ai acumulatorului;
- P (paritate) – setat dacă suma modulo 2 a acumulatorului este 1.

Cele patru instrucţiuni aritmetice sunt:

- Adunare:
  - INC – adună 1 la operandul sursă. Rezultatul este returnat în operand.
  - ADD – adună acumulatorul cu operandul sursă. Rezultatul este returnat în acumulator.
  - ADDC - adună acumulatorul cu operandul sursă şi bitul C. Rezultatul este returnat în acumulator.
  - DA – realizează o corecţie a adunării în situaţia lucrului cu numere codificate BCD.
- Scădere:
  - DEC – scade 1 din operandul sursă. Rezultatul este returnat în operand.
  - SUBB – scade din acumulator operandul sursă şi bitul C. Rezultatul este returnat în acumulator.
- Înmulţire:
  - MUL – realizează o înmulţire fără semn între registrele A şi B. Rezultatul este returnat în registrele A (octetul mai puţin semnificativ) şi B (octetul mai semnificativ).
- Împărţire:
  - DIV – realizează o împărţire fără semn între registrele A şi B. Câtul rezultatului este păstrat în registrul A iar restul împărţirii în B.

c) Instrucţiuni logice

Arhitectura familiei 8xC552 care execută operaţii logice poate fi considerată un *procesor boolean* de sine stătător. Astfel acesta dispune de instrucţiuni proprii, acumulator (bitul C din PSW), execută instrucţiuni pe un bit, 4 biţi sau un octet şi poate adresa direct locaţii de memorie sau porturile de intrare-ieşire.

Instrucţiunile logice sunt:

CLR – şterge conţinutul acumulatorului sau al altui registru adresat direct.
SETB – setează bitul C sau alt bit adresat direct.

CPL – complementează (în complement faţă de 1) acumulatorul sau orice bit adresabil direct.

RL – roteşte la stânga conţinutul acumulatorului. Este echivalent cu o înmulţire cu 2.

RR – roteşte la dreapta conţinutul acumulatorului. Este echivalent cu o împărţire cu 2.

RLC – roteşte la stânga conţinutul acumulatorului prin intermediul bitului C care devine egal cu ultimul bit deplasat.

RRC – roteşte la dreapta conţinutul acumulatorului prin intermediul bitului C care devine egal cu ultimul bit deplasat.

SWAP – roteşte la stânga de 4 ori conţinutul acumulatorului. Este echivalent cu o schimbare a biţilor 0-3 cu 4-7.

ANL – execută o operaţie ŞI între doi operanzi. Rezultatul este returnat în primul operand.

ORL – execută o operaţie SAU între doi operanzi. Rezultatul este returnat în primul operand.

XRL – execută o operaţie SAU exclusiv între doi operanzi. Rezultatul este returnat în primul operand.

d) Controlul programului.

Instrucţiunile pentru controlul programului determină, uneori concomitent cu îndeplinirea unei condiţii, executarea nesecvenţială a instrucţiunilor din program.

Există trei clase de astfel de instrucţiuni: apel necondiţionat la subrutină, întoarcere din subrutină şi salt; salturi condiţionale; revenire din întrerupere.

- Apeluri şi salturi necondiţionate:
  - ACALL– memorează următoarea adresă în stivă, incrementează SP cu 2 şi apoi transferă controlul adresei de salt; ACALL poate fi utilizat pentru apeluri într-o pagină de memorie de 2 kB (locaţia de salt este adresată pe 11 biţi).
  - LCALL– memorează următoarea adresă în stivă, incrementează SP cu 2 şi apoi transferă controlul adresei de salt; LCALL poate fi utilizat pentru apeluri într-o pagină de memorie de 64 kB.
  - AJMP – execută un salt la adresa specificată; AJMP poate fi utilizat pentru salturi într-o pagină de memorie de 2 kB (locaţia de salt este adresată pe 11 biţi).
  - LJMP – execută un salt la adresa specificată; LJMP poate fi utilizat pentru apeluri într-o pagină de memorie de 64 kB.
  - SJMP – execută un salt la adresa specificată; SJMP poate fi utilizat pentru apeluri într-o pagină de memorie de 256 octeţi. SJMP este un salt relativ faţă de adresa de pornire în domeniul (-128…+127).

- JMP – execută un salt relativ la conţinutul registrului DPTR, ca deplasament folosind valoarea din acumulator.
- RET – transferă controlul adresei de salt salvate în stivă de o instrucţiune CALL anterioară şi decrementează SP cu 2.

- Salturi condiţionate:
  - JZ – execută salt dacă acumulatorul este zero. Saltul este relativ faţă de adresa curentă.
  - JNZ – execută salt dacă acumulatorul este diferit de zero. Saltul este relativ faţă de adresa curentă.
  - JC – execută salt dacă bitul C este 1. Saltul este relativ faţă de adresa curentă.
  - JNC – execută salt dacă bitul C este 0. Saltul este relativ faţă de adresa curentă.
  - JB – execută salt dacă bitul operandului este 1. Saltul este relativ faţă de adresa curentă.
  - JNB – execută salt dacă bitul operandului este 0. Saltul este relativ faţă de adresa curentă.
  - CJNE – compară primul operand cu al doilea şi execută salt dacă nu sunt egali.
  - DJNZ – decrementează operandul sursă şi execută salt dacă rezultatul este diferit de zero.
- Revenire din întrerupere:
  - RETI – transferă controlul într-o manieră asemănătoare cu RET, cu deosebirea că această instrucţiune validează întreruperile pentru nivelul curent de priorităţi.

1.4. Porturile de intrare-ieşire

Controlerul 80C552 are 6 porturi de 8 biţi pentru intrări/ieşiri digitale. Fiecare port constă într-un registru (registrele speciale P0-P5), un amplificator tampon de intrare (buffer de intrare) şi un buffer de ieşire (numai porturile P0 şi P4). Porturile P0-P3 sunt identice cu cele ale lui 8051, cu excepţia funcţiilor adiţionale ale portului P1 (liniile P1.6 şi P1.7 pot fi setate ca linii de comunicaţie pentru interfaţa serială sincronă I^2C; acest lucru implică un buffer de ieşire de tip *drenă în gol*) şi a porturilor suplimentare P4 şi P5: portul P4 are o funcţionalitate asemănătoare cu a porturilor P0-P3; portul P5 poate fi utilizat numai ca port de intrare.

Figura 1.3 arată structura funcţională a porturilor de intrare-ieşire a circuitului 80C552. Un bit al registrului portului corespunde unui bit din registrul special al portului şi constă într-un bistabil D.

Pentru porturile P0-P4 fiecare linie poate fi configurată ca linie de intrare sau de ieşire. În situaţia configurării ca linie de intrare, amplificatorul de ieşire trebuie blocat, blocare realizată prin setarea portului. Această

procedură nu trebuie aplicată decât dacă portul a fost folosit ca port de ieşire, întrucât iniţial, după RESET, toate registrele porturilor P0-P4 sunt încărcate cu FFh.

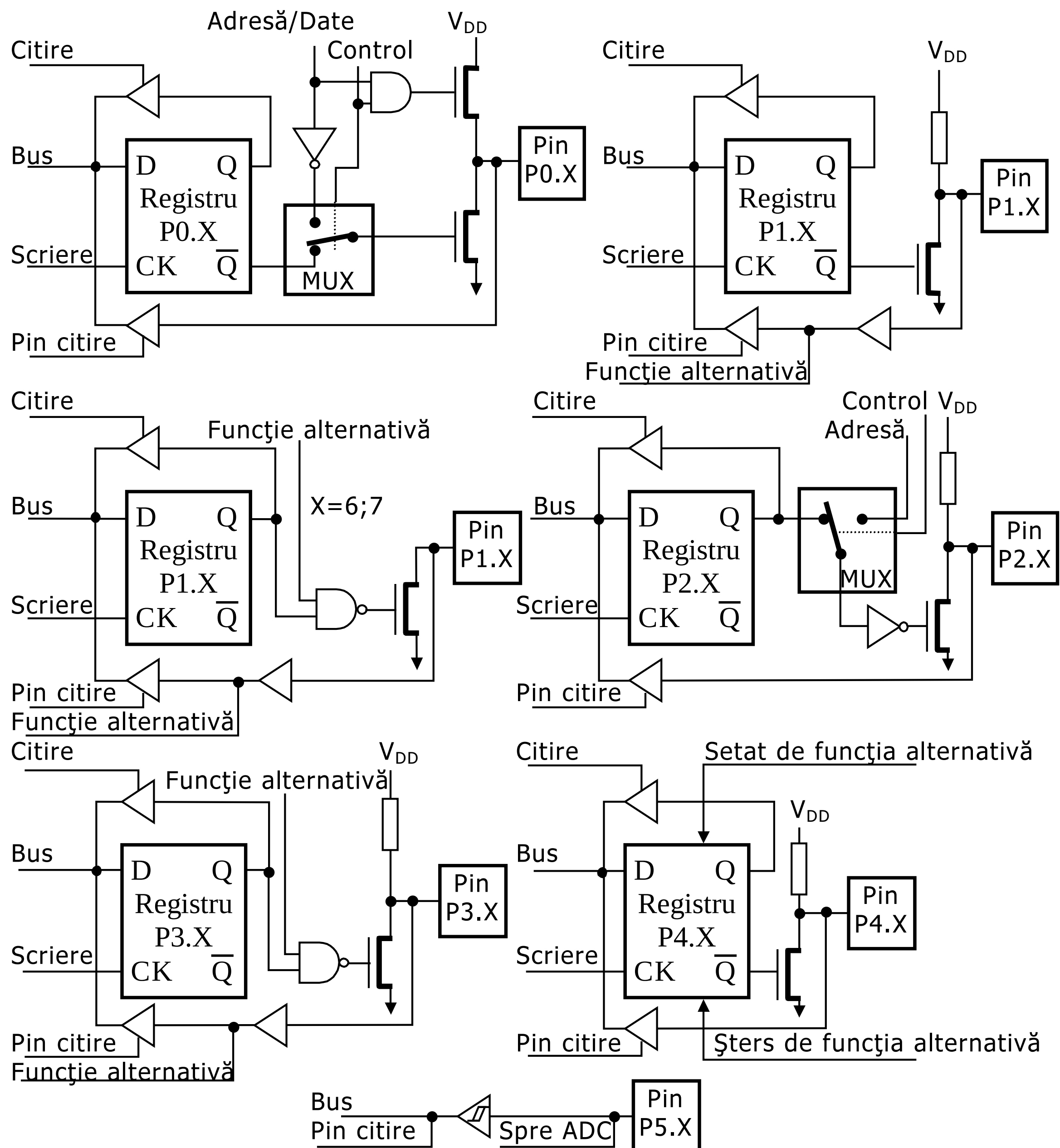


Figura 1.3. Porturile de intrare-ieşire

Portul P0, în situaţia controlerului 80C552, este folosit numai ca magistrală multiplexată de adrese (octetul inferior) şi date.

Portul P2 poate fi utilizat ca octet superior pentru adresarea memoriei externe pe 16 biţi. Dacă este folosită adresarea pe 8 biţi, este implicat numai portul P0 şi atunci P2 devine port de uz general.

Portul P5 este port numai de intrare şi poate utiliza liniile lăsate libere de convertorul analog-numeric.

Amplificatoarele de ieşire ale porturilor P2, P3, P4 şi P1.0-P1.5 sunt capabile să conducă 4 intrări compatibile LSTTL. Folosit ca port de ieşire, P0 are o capabilitate de 8 intrări LSTTL, dar necesită rezistenţe externe deoarece ieşirile sunt cu drenă în gol.

Citirea porturilor, aşa cum se observă şi din figura 1.3. poate fi făcută fie citind direct pinul, fie citind bistabilul portului respectiv. Selectarea modului de citire este făcută automat de unitatea centrală, funcţie de instrucţiune.

Funcţiile alternative ale porturilor sunt descrise în figura 1.1 şi tabelul 1.4, detalii suplimentare despre acestea găsindu-se la descrierea modulelor interne.

Tabelul 1.4

P0.0	AD0	Magistrală multiplexată date şi adrese (octetul mai puţin semnificativ) pe durata accesului la memoria externă	P1.0	CT0I	Semnale captură pentru timer T2
P0.1	AD1		P1.1	CT1I	
P0.2	AD2		P1.2	CT2I	
P0.3	AD3		P1.3	CT3I	
P0.4	AD4		P1.4	T2	Ieşire timer T2
P0.5	AD5		P1.5	RT2	Reset extern timer T2
P0.6	AD6		P1.6	SCL	Ceas interfaţă serială I^2C
P0.7	AD7		P1.7	SDA	Date interfaţă serială I^2C
P2.0	A9	Magistrală de adrese (octetul mai semnificativ) pe durata accesului la memoria externă	P3.0	RXD	Intrare date UART
P2.1	A9		P3.1	TXD	Ieşire date UART
P2.2	A10		P3.2	$\overline{INT0}$	Întrerupere externă 0
P2.3	A11		P3.3	$\overline{INT1}$	Întrerupere externă 1
P2.4	A12		P3.4	T0	Intrare externă timer T0
P2.5	A13		P3.5	T1	Intrare externă timer T1
P2.6	A14		P3.6	$\overline{WR}$	Scriere memorie externă
P2.7	A15		P3.7	$\overline{RD}$	Citire memorie externă
P4.0	CMSR0	Comparare cu timer T2 şi setări/resetări ieşiri	P5.0	ADC0	Opt intrări analogice pentru convertorul analog numeric
P4.1	CMSR1		P5.1	ADC1	
P4.2	CMSR2		P5.2	ADC2	
P4.3	CMSR3		P5.3	ADC3	
P4.4	CMSR4		P5.4	ADC4	
P4.5	CMSR5		P5.5	ADC5	
P4.6	CMT0	Comparare cu timer T2 şi comutări ieşiri	P5.6	ADC5	
P4.7	CMT1		P5.7	ADC5	

1.5. Modulatorul de impulsuri în durată

Familia 8xC552 conţine două modulatoare de impulsuri în durată, codificate PWM0, respectiv PWM1. Impulsurile generate de aceste modulatoare au controlate independent durata şi perioada de repetiţie.

Structura internă a modulatorului de impulsuri al circuitului 8xC552 este prezentat în figura 1.4.

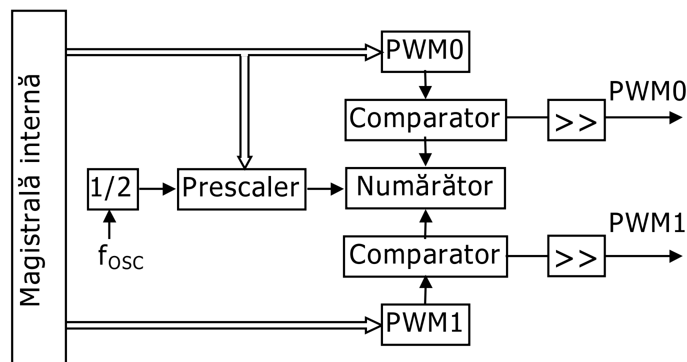


Figura 1.4. Modulatorul de impulsuri în durată

Frecvenţa de repetiţie pentru ambele modulatoare este stabilită de un prescaler care furnizează frecvenţa de ceas pentru numărător. Coeficientul de divizare al prescalerului este definit de registrul special PWMP.

Perioada de repetiţie a modulatorului PWM este dată de relaţia:

$$f_{PWM} = \frac{f_{OSC}}{2 \cdot (1 + PWMP) \cdot 255}$$

şi, pentru un oscilator de 12 MHz semnifică frecvenţe de repetiţie de la 92 Hz la 23.5 kHz.

Valoarea contorului numărătorul de 8 biţi (între 0 şi 254) este examinată de două comparatoare comandate de câte un registru special pentru fiecare modulator: PWM0, respectiv PWM1.

Coeficientul de umplere stabilit de cele două registre se determină cu formula:

$$\text{coef. umplere} = \frac{PWMx}{255 - PWMx}$$

Această structură este capabilă să asigure celor două ieşiri impulsuri cu coeficienţi de umplere între 0 şi 1, în incremente de 1/255.

1.6. Convertorul analog numeric

Modulul analogic constă într-un multiplexor analogic cu opt intrări, un registru de aproximaţii succesive de 10 biţi, un comparator, un convertor numeric/analogic şi logica de comandă a acestor blocuri. Tensiunea de referinţă şi alimentările sunt asigurate din exterior, pe pini separaţi.

O conversie durează 50 de cicluri maşină iar semnalul analogic de intrare trebuie să îndeplinească condiţia: $0 \le AV_{SS} \le V_{REF-} \le \text{Semnal} \le V_{REF+} \le AV_{DD} \le +5V$, unde AV_{SS} şi AV_{DD} reprezintă tensiunile de alimentare analogice iar $V_{REF\pm}$ tensiunile de referinţă.

Controlul convertorului analog/numeric este asigurat de registrul special ADCON, prezentat în tabelul 1.5.

Structura funcţională a modulului analogic este prezentată în figura 1.5.

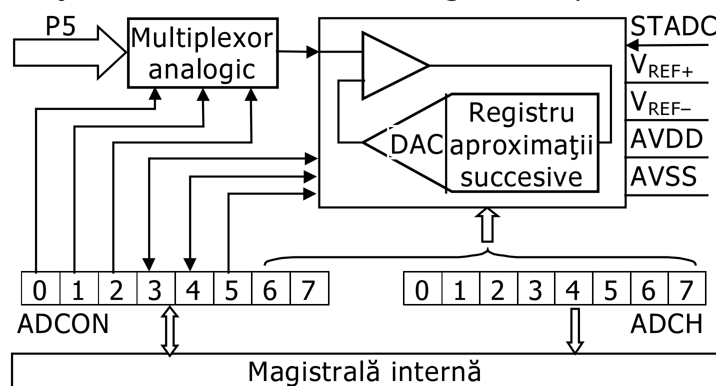


Figura 1.5. Convertorul analog/numeric

Rezultatul conversiei este regăsit în registrul ADCH (biţii mai semnificativi), ultimii doi biţi fiind citiţi din registrul ADCON (ADC.1 şi ADC.0).

Cât timp o conversie este în curs (ADCI şi ADCS diferite de zero), un semnal extern STADC sau o comandă software nu este luată în consideraţie şi nu este iniţializată o nouă conversie.

Tabelul 1.5

ADCON (C5h)	ADC.1	ADC.0	ADEX	ADCI	ADCS	AADR2	AADR1	AADR0

ADC.1	Bitul 1 al rezultatului conversiei
ADC.0	Bitul 0 al rezultatului conversiei
ADEX	Validare conversie externă (pin ADCS): 0: conversia este iniţializată numai prin program (bitul ADCS) 1: conversia este iniţializată prin program sau hardware.
ADCI	Indicator întrerupere ADC: - dacă este setat, rezultatul conversiei poate fi citit; - dacă este validat, sfârşitul conversiei poate genera o întrerupere; - cât timp este setat nu se poate declanşa o altă conversie. Acest bit este "numai citire".
ADCS	Starea convertorului ADC. Poate fi setat prin program sau hardware (semnalul STADC). Semnificaţia sa este: (vezi tabelul de mai jos)
AADR2 AADR1 AADR0	Comanda multiplexorului analogic. Selectarea intrărilor portului P5 este: (vezi tabelul de mai jos)

ADCI	ADCS	Stare convertor
0	0	se poate iniţia conversie; ADC neocupat.
0	1	nu poate fi iniţiată o nouă conversie.
1	0	nu poate fi iniţiată o altă conversie.
1	1	rezervat

AADR2	AADR1	AADR0	Intrare analogică
0	0	0	ADC.0 (P5.0)
0	0	1	ADC.1 (P5.1)
0	1	0	ADC.2 (P5.2)
0	1	1	ADC.3 (P5.3)
1	0	0	ADC.4 (P5.4)
1	0	1	ADC.5 (P5.5)
1	1	0	ADC.6 (P5.6)
1	1	1	ADC.7 (P5.7)

Valoarea numerică a tensiunii analogice convertite este dată de relaţia:

$$V_{ADC} = 1024 \cdot \frac{V_{IN} - V_{REF-}}{V_{REF+} - V_{REF-}}$$

Este recomandabil ca toate tensiunile analogice (AV_{SS}, AV_{DD} şi $AV_{REF\pm}$) să fie asigurate de o sursă de alimentare separată.

1.7. Timere/Numărătoare

Familia 8xC552 dispune de patru timere: timerele T0 şi T1 sunt identice cu cele din 8051, T2 este un timer de 16 biţi cu facilităţi suplimentare iar T3 este un timer de 8 biţi pentru resetarea programului controlerului.

Timerele 0 şi 1 constau în două numărătoare de 16 biţi şi pot îndeplini următoarele funcţii:

- măsurarea unor intervale de timp;
- numărarea unor evenimente;
- generarea unor cereri de întrerupere.

Configurat ca timer, contorul este incrementat la fiecare ciclu maşină (la fiecare microsecundă pentru oscilator de 12 MHz). Astfel, timpul poate fi măsurat în unităţi de cicluri maşină.

Configurat ca numărător, contorul este incrementat la fiecare tranziţie $1 \rightarrow 0$ a semnalului de pe pinul de intrare corespunzător. Deoarece o recunoaştere a unei tranziţii durează două cicluri maşină, frecvenţa maximă de numărare este de 500 kHz (pentru oscilator de 12 MHz).

Faţă de aceste moduri, timerele T0 şi T1 mai au patru moduri de funcţionare, de la MOD0 la MOD3.

Selectarea modului de lucru şi controlul timerelor T0 şi T1 sunt asigurate de registrele speciale TMOD, respectiv TCON descrise în tabelul 1.6.

Tabelul 1.6

TMOD (89h)	Timer 1				Timer 0			
	GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0

GATE	Dacă este setat, timerul x funcţionează dacă pinul INTx este la 1 logic şi bitul TRx (din registrul TCON) este setat. Dacă este şters, timerul este condiţionat numai de starea bitului TR.
C/$\overline{T}$	Dacă este setat, modulul funcţionează ca numărător (numără tranziţiile 1→0 ale pinului Tn). Dacă este şters funcţionează ca timer, numărând ciclurile maşină.

		M1	M0	Mod	Descriere
M1, M0	Selecţie mod:	0	0	0	Numărător de 8 biţi cu prescaler divizor cu 32
		0	1	1	Numărător de 16 biţi
		1	0	2	Numărător de 8 biţi cu reîncărcare automată
		1	1	3	T0: numărător de 8 biţi; T1: oprit.

TCON (88h)	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TFx	Indicator depăşire timer Tx. El este setat în momentul depăşirii şi este şters automat în momentul în care unitatea centrală dă controlul rutinei de întrerupere.
TRx	Comandă funcţionarea timerului Tx. Dacă este setat sau şters, timerul funcţionează, respectiv este oprit.
IEx	Indicator pentru frontul întreruperii externe. Dacă ITx este setat, o tranziţie 1→0 a intrării INTx va seta acest bit. Bitul este şters automat în momentul în care unitatea centrală dă controlul rutinei de întrerupere.
ITx	Stabileşte dacă întreruperea externă acţionează pe front sau pe nivel. Dacă este setat, întreruperea externă este activă pe frontul descrescător al semnalului de pe pinul INTx; dacă este şters, întreruperea externă este activă pe nivelul zero logic al pinului INTx.

Semnificaţia celor patru moduri de funcţionare este prezentată în paragrafele următoare.

1.7.1 Modul 0

În acest mod timerele T0 sau T1, sunt configurate ca registre de 13 biţi. În momentul în care valoarea numărătorului x trece de la 1FFFh la 0h este setat automat bitul TFx. Intrarea în numărător este validată dacă bitul TRx este setat iar bitul GATE şi intrarea INTx respectă condiţiile descrise anterior.

Registrul de 13 biţi este format din 8 biţi ai registrului THx şi 5 biţi ai registrului TLx. Cei 3 biţi mai semnificativi ai lui TLx sunt nedeterminaţi şi trebuie ignoraţi.

1.7.2 Modul 1

Este identic cu modul 0, numai că registrul de numărare este de 16 biţi.

1.7.3 Modul 2

Acest mod configurează timerele T0 sau T1 ca două numărătoare de 8 biţi, având facilitatea de reîncărcare: în momentul în care TLx trece de la FFh la 0h, nu se setează numai bitul TFx, dar TLx este încărcat cu valoarea lui THx. THx rămâne nemodificat.

1.7.4 Modul 3

În acest mod, timerul T0 funcţionează ca două registre separate, TL0 şi TH0, fiecare a câte 8 biţi. Registrul TL0 este controlat de biţii standard ai timerulului T0 iar registrul TH0 de biţii timerului T1, inclusiv întreruperea timerului T1.

Modul 3 este adoptat dacă este necesar un timer suplimentar. În această situaţia T0 lucrează ca două timere independente iar T1 poate funcţiona, de exemplu, ca generator de rată de transmisie sau în orice altă aplicaţie care nu necesită o întrerupere.

Configuraţia internă a timerelor T0 şi T1 în cele patru moduri de funcţionare se prezintă în figura 1.6.

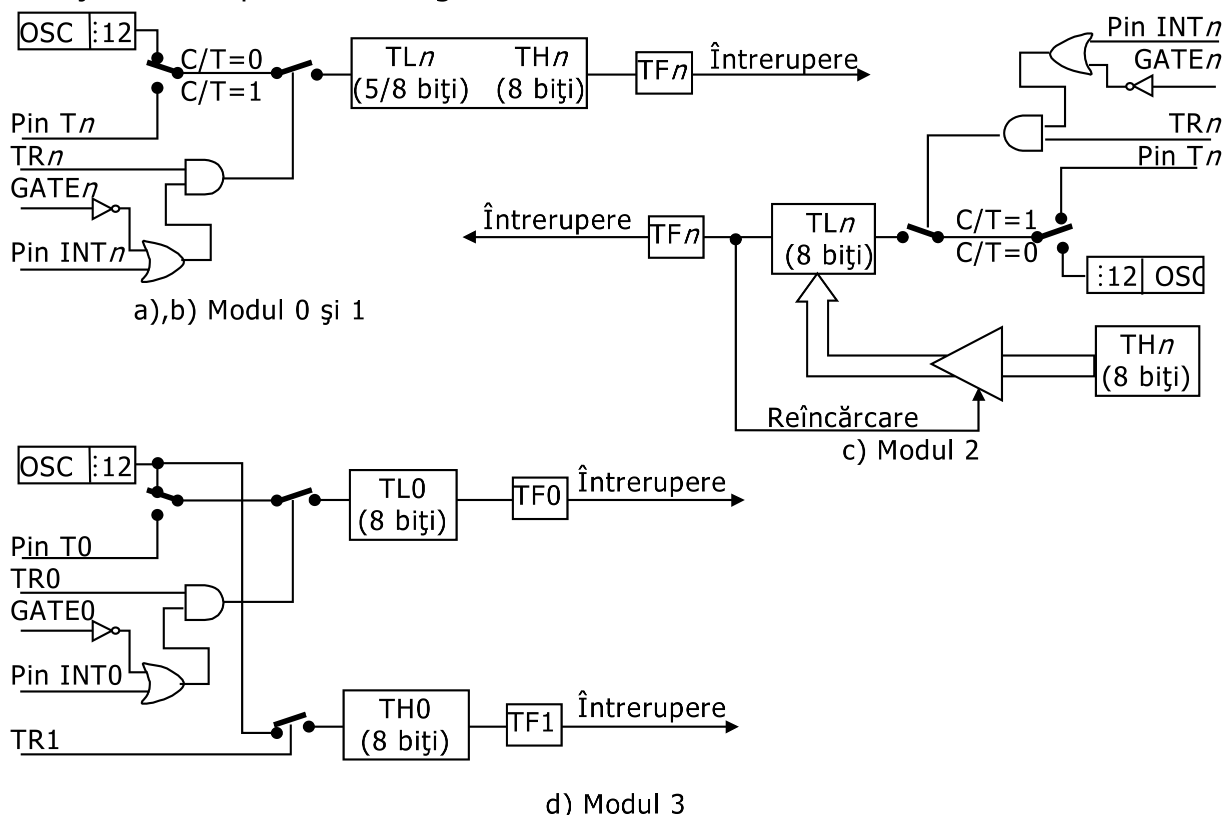


Figura 1.6. Modurile de lucru al timerelor T0 şi T1

Întreruperile generate de timerele T0 şi T1 sunt gestionate de biţii ET0, respectiv ET1 aflate în registrul special pentru controlul întreruperilor (IEN0.1, respectiv IEN0.3).

Timerul T2 este un numărător de 16 biţi conectat la patru registre de captură de 16 biţi şi trei registre de comparare de 16 biţi. Registrele de captură sunt folosite pentru a înregistra conţinutul registrului timerului T2 în momentul în care survine o tranziţie pe pinul corespunzător (CT0I, CT1I, CT2I şi CT3I). Registrele de comparare sunt folosite pentru a seta, şterge sau comuta biţii corespunzători din portul P4 atunci când conţinutul registrului T2 este atinge valorile din cele trei registre.

Timerul T2 este format din două registre de 8 biţi TMH2 (octetul mai semnificativ), respectiv TML2 (octetul mai puţin semnificativ).

Frecvenţa de numărare poate fi furnizată de oscilatorul intern (divizat cu 12) sau de la o sursă externă (un front crescător pe pinul T2 – P1.4), ulterior aceste frecvenţe fiind divizate într-un prescaler programabil cu 1, 2, 4 sau 8.

În situaţia folosirii timerului T2 ca numărător, frecvenţa maximă de lucru este dublă faţă de timerele T0 şi T1, respectiv de 1MHz.

Valoarea timerului poate fi citită fără ca acesta să fie oprit dar, acesta neavând registre suplimentare de citire, trebuie luate măsuri de precauţie în situaţia apariţiei unei depăşiri a registrului chiar în timpul citirii.

Registrul timerului T2 nu poate fi scris ci numai şters prin intermediul unor semnale externe: RST sau un front crescător pe pinul RT2 (validarea ştergerii pe pinul RT2 este produsă prin setarea bitului T2ER–TM2CON.5).

Pot fi generate cereri de întreruperi pentru depăşiri ale registrului TM2, atât la nivel de 8 biţi (depăşirea registrului TML2), cât şi la 16 biţi (depăşirea registrului TMH2); în ambele situaţii, vectorul de întrerupere este acelaşi. În situaţia depăşirii registrului TML2 este setat indicatorul TLBO (din registrul special TM2CON) iar când se produce depăşirea registrului TMH2 sunt setaţi indicatorii T2OV şi TLBO. Pentru validarea întreruperii generate de TMH2 este necesară setarea biţilor ET2 (IEN1.7) şi T2IS0 (TM2CON.4). Validarea întreruperii produse de TMH2 este făcută prin setarea biţilor ET2 (IEN1.7) şi T2IS1 (TM2CON.7). Indicatorii setaţi de depăşire trebuie şterşi prin programul de tratare a întreruperii.

Timerul T2 are în structură patru registre de captură: CT0, CT1, CT2 şi CT3. Aceste registre de 16 biţi sunt încărcate funcţie de semnalele externe CT0I, CT1I, CT2I şi CT3I. Funcţie de starea registrului special TM2IR, o dată cu aplicarea semnalelor CTxI se generează şi o întrerupere. Conţinutul registrului de control al capturii CTCON oferă posibilitatea selectării modului de acţiune al semnalelor CTxI: pe front crescător, pe front descrescător sau pe ambele tipuri de fronturi.

Timerul TM2 mai conţine şi trei registre de comparare CM0, CM1 şi CM2. Conţinutul acestor registre este verificat la fiecare incrementare a numărătorului T2. Atunci când unul din registrele CMx are aceeaşi valoare cu registrul TM2, este setat bitul corespunzător din registrul special TM2IR. Suplimentar, registrele de comparare mai pot comanda şi biţii portului P4 funcţie de registrele speciale STE şi RTE. Astfel, CM0 poate seta biţii 0-5 ai P4 dacă sunt setaţi biţii corespunzători din registrul STE; CM1 şterge biţii 0-5 ai P4 dacă sunt setaţi biţii corespunzători din registrul RTE; biţii 6 şi 7 ai P4 sunt inversaţi de CM2 dacă este setat corespunzător registrul RTE.

Structura internă a timerului T2 şi a registrelor de captură şi comparare este prezentată în figura 1.7.

Timerul T2 poate genera nouă întreruperi. Opt indicatoare de întreruperi se găsesc în registrul TM2IR iar al nouălea este bitul TM2CON.4. Priorităţile întreruperilor generate de TM2 sunt controlate de registrul IP1.

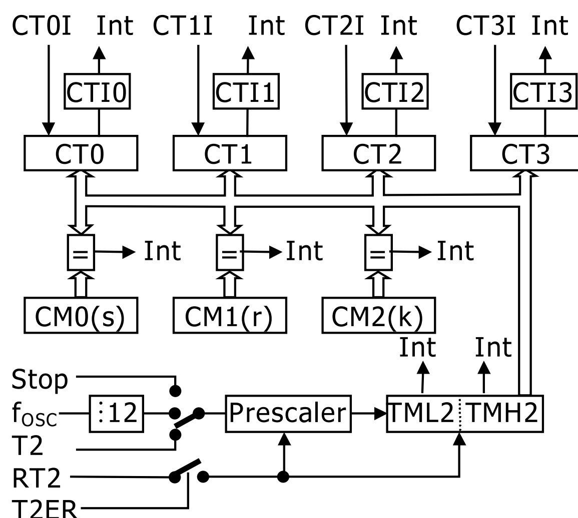


Figura 1.7. Timerul T2

În concluzie, registrele speciale folosite de timerul T2 sunt TM2CON, CTCON, STE, RTE, TM2IR, IEN1 şi IP1. Conţinutul acestora la nivel de bit este prezentat în tabelul 1.7.

								Tabelul 1.7
TM2CON (EAh)	T2IS1	T2IS0	T2ER	T2BO	T2P1	T2P0	T2MS1	T2MS0
T2IS1 T2IS0 T2ER T2BO	Selectare întrerupere depăşire 16 biţi. Selectare întrerupere depăşire 8 biţi. Validare resetare externă; T2 poate fi şters de un front crescător pe pinul RT2 (P1.5). Indicator întrerupere depăşire 8 biţi.							
T2P1 T2P0	Selectare valoare prescaler.		0 0 1 1	0 1 0 1	:1 :2 :4 :8			
T2MS1 T2MS0	Selectare mod de lucru.		0 0 1 1	0 1 0 1	Oprit Mod timer (f_{OSC}/12) Rezervat Mod numărător (pin T2)			
CTCON (EBh)	CTN3	CTP3	CTN2	CTP2	CTN1	CTP1	CTN0	CTP0
CTN3 CTP3 CTN2 CTP2 CTN1 CTP1 CTN0 CTP0	Registrul captură 3 comandat de front căzător pe pinul CT3I Registrul captură 3 comandat de front crescător pe pinul CT3I Registrul captură 2 comandat de front căzător pe pinul CT2I Registrul captură 2 comandat de front crescător pe pinul CT2I Registrul captură 1 comandat de front căzător pe pinul CT1I Registrul captură 1 comandat de front crescător pe pinul CT1I Registrul captură 0 comandat de front căzător pe pinul CT0I Registrul captură 0 comandat de front crescător pe pinul CT0I							
STE (EEh)	TG47	TG46	SP45	SP44	SP43	SP42	SP41	SP40

TG47	Dacă sunt setaţi, la comutarea comparatorului CM2, biţii corespunzători din P4 sunt setaţi.
TG46	Dacă sunt şterşi, la comutarea comparatorului CM2, biţii corespunzători din P4 sunt şterşi.
SP45	Dacă este setat, P4.5 va deveni 1 logic când CM0=TM2.
SP44	Dacă este setat, P4.4 va deveni 1 logic când CM0=TM2.
SP43	Dacă este setat, P4.3 va deveni 1 logic când CM0=TM2.
SP42	Dacă este setat, P4.2 va deveni 1 logic când CM0=TM2.
SP41	Dacă este setat, P4.1 va deveni 1 logic când CM0=TM2.
SP40	Dacă este setat, P4.0 va deveni 1 logic când CM0=TM2.

RTE (EFh)	TP47	TP46	RP45	RP44	RP43	RP42	RP41	RP40

TP47	Dacă sunt setaţi, la comutarea comparatorului CM2, bitul din portul P4 este inversat.
TP46	Dacă este setat, P4.5 va deveni 0 logic când CM1=TM2.
RP45	Dacă este setat, P4.4 va deveni 0 logic când CM1=TM2.
RP44	Dacă este setat, P4.3 va deveni 0 logic când CM1=TM2.
RP43	Dacă este setat, P4.2 va deveni 0 logic când CM1=TM2.
RP42	Dacă este setat, P4.1 va deveni 0 logic când CM1=TM2.
RP41	Dacă este setat, P4.0 va deveni 0 logic când CM1=TM2.
RP40	

TM2IR (C8h)	T2OV	CMI2	CMI1	CMI0	CTI3	CTI2	CTI1	CTI0

T2OV	Indicator întrerupere CM2.
CMI2	Indicator întrerupere CM1.
CMI1	Indicator întrerupere CM0.
CMI0	Indicator întrerupere CT3.
CTI3	Indicator întrerupere CT2.
CTI2	Indicator întrerupere CT1.
CTI1	Indicator întrerupere CT0.
CTI0	

IEN1 (E8h)	ET2	ECM2	ECM1	ECM0	ECT3	ECT2	ECT1	ECT0

ET2	Validare întrerupere CM2.
ECM2	Validare întrerupere CM1.
ECM1	Validare întrerupere CM0.
ECM0	Validare întrerupere CT3.
ECT3	Validare întrerupere CT2.
ECT2	Validare întrerupere CT1.
ECT1	Validare întrerupere CT0.
ECT0	

IP1 (F8h)	PT2	PCM2	PCM1	PCM0	PCT3	PCT2	PCT1	PCT0

PT2	Nivel prioritate depăşire numărare.
PCM2	Nivel prioritate CM2.
PCM1	Nivel prioritate CM1.
PCM0	Nivel prioritate CM0.
PCT3	Nivel prioritate CT3.
PCT2	Nivel prioritate CT2.
PCT1	Nivel prioritate CT1.
PCT0	Nivel prioritate CT0.
	Dacă bitul este setat se selectează nivelul superior de prioritate.

1.8. Timerul iniţializare T3 (watchdog)

Acest timer este destinat iniţializării periodice a controlerului în situaţia existenţei unei erori de program (buclă infinită) sau apariţiei unor perturbaţii externe care pot comanda aleator procesorul. Dacă este validat, timerul T3 generează resetul sistemului dacă programul nu reiniţializează periodic conţinutul acestui timer.

Timerul T3 constă într-un numărător de 8 biţi şi un divizor de 11 biţi; prescalerul divizează o frecvenţă de 1 MHz (în situaţia utilizării unui oscilator de 12 MHz).

Dacă numărătorul de 8 biţi are o tranziţie FFh→0h, este produs un impuls care resetează controlerul. De asemenea, prin intermediul unui buffer, semnalul este scos şi pe pinul RST unde poate fi utilizat pentru iniţializarea circuitelor de suport. Deoarece este un impuls scurt (circa 3 μs), existenţa unei capacităţi în circuitul extern de reset face semnalul produs de controler inutilizabil.

Perioada de repetiţie a timerului T3 este dictată de valoarea cu care este reîncărcat: perioada minimă este de 2 ms, dacă timerul este încărcat cu FFh; perioada maximă de repetiţie este de 512 ms şi corespunde unei valori de reîncărcare de 0h. Relaţia pentru determinarea perioadei de repetiţie este:

$$T_R = \frac{24576 \cdot (T3+1)}{f_{OSC}} \quad [s]$$

Pentru iniţializarea automată a programului, este necesară mai întâi, determinarea timpului de execuţie al programului, după care timerul T3 este programat cu o valoare acoperitoare. Dacă timpul de execuţie este mai mare de 512 ms, este necesară partiţionarea programului în mai multe module, fiecare modul asigurând o reprogramare a timerului T3.

Pentru a preveni erori de program la reîncărcarea sa, timerul T3 este programat în doi paşi: mai întâi, este setat bitul WLE (PCON.4), apoi T3 poate fi încărcat cu valoarea dorită. După încărcare, WLE este şters automat.

Suplimentar, modulul de iniţializare este condiţionat şi de semnalul extern $\overline{EW}$. Când $\overline{EW}$ are valoarea 0 logic, funcţionarea timerului T3 nu se poate dezactiva prin program.

Schema bloc a modului timer T3 este indicată în figura 1.8.

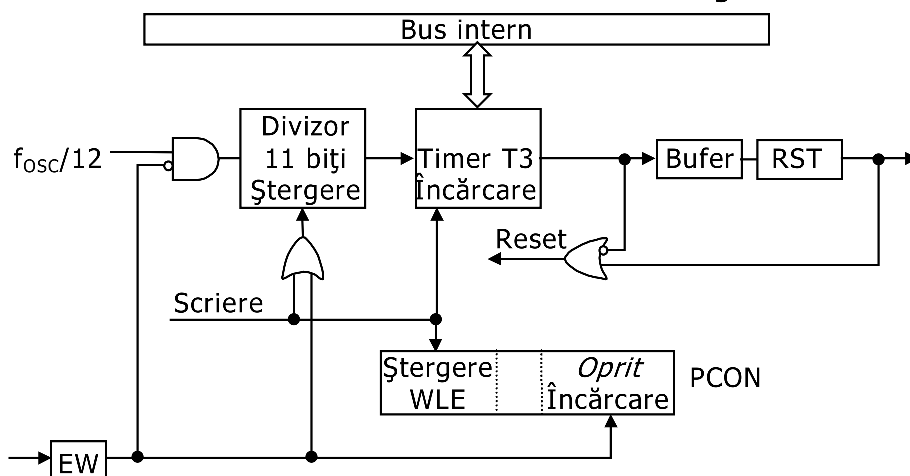


Figura 1.8. Timerul T3

În modurile de funcţionare cu consum redus de energie, IDLE şi POWER-DOWN, funcţionarea timerului T3 poate pune probleme. Astfel, dacă în modul IDLE timerul este funcţional, în modul POWER-DOWN funcţionarea timerului este contradictorie cu modul de lucru ales. Pentru a preveni acest lucru, semnalul extern $\overline{EW}$ nu lasă funcţionale simultan cele două module.

Bitul de control al timerului T3, WLE, va fi prezentat la paragraful 1.12.1. unde este descris registrul special PCON.

1.9. Interfaţa serială asincronă

Această interfaţă serială, cunoscută şi sub numele de SIO0, poate transmite şi recepţiona date simultan, adică este o interfaţă full duplex. De asemenea, registrul de recepţie este bufferat: modulul poate iniţia o recepţie a unui nou octet chiar dacă octetul anterior nu a fost citit din registrul de recepţie. Suplimentar faţă de interfaţa serială a circuitelor 8051 are câteva facilităţi suplimentare:

- protecţia la erori de încadrare (*framing error*) prin intermediul bitului de stop;
- protecţia la erori de paritate, fiind posibilă transmisia bitului de paritate;
- are un mod de lucru destinat comunicaţiei multiprocesor.

Interfaţa serială asincronă poate opera în patru moduri.

a) modul 0;
b) modul 1;
c) modul 2;
d) modul 3.

Controlul interfeţei seriale este făcut de registrul special S0CON, cu structura prezentată în tabelul 1.8.

Tabelul 1.8

S0CON (98H)	SM0	SM1	SM2	REN	TB8	RB8	T1	R1

<table>
<tr><td rowspan="5">SM0
SM1</td><td rowspan="5">Selectare mod lucru:</td><td>SM0</td><td>SM1</td><td>Mod</td><td>Descriere</td><td>Viteză de transmisie</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Registru de deplasare</td><td>$f_{OSC}/12$</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>UART de 8 biţi</td><td>Variabilă</td></tr>
<tr><td>1</td><td>0</td><td>2</td><td>UART de 9 biţi</td><td>$f_{OSC}/32$ sau $f_{OSC}/64$</td></tr>
<tr><td>1</td><td>1</td><td>3</td><td>UART de 9 biţi</td><td>Variabilă</td></tr>
<tr><td>SM2</td><td colspan="6">Folosit pentru comunicarea multiprocesor în modurile 2 şi 3. Dacă este setat, RI nu va fi activat dacă bitul 8 de date nu este 0 logic. În modul 1 condiţionează RI de primirea unui bit valid de stop. În modul 0 trebuie şters.</td></tr>
<tr><td>REN</td><td colspan="6">Validarea recepţiei seriale. Se poate seta prin program pentru a activa sau dezactiva recepţia serială.</td></tr>
<tr><td>TB8</td><td colspan="6">Bitul 9 de date transmis în modul 2 sau 3. Este setat sau şters prin program de regulă, funcţie de bitul de paritate.</td></tr>
<tr><td>RB8</td><td colspan="6">Bitul 9 de date recepţionat în modul 2 sau 3. În modul 1, dacă SM2=0, constituie bitul de stop. În modul 0 nu este folosit.</td></tr>
<tr><td>TI</td><td colspan="6">Indicator întrerupere transmisie. Este setat automat la terminarea mesajului. Trebuie şters prin program.</td></tr>
<tr><td>RI</td><td colspan="6">Indicator întrerupere recepţie. Este setat automat la recepţionarea unui mesaj. Trebuie şters prin program</td></tr>
</table>

1.9.1 Interfaţa serială SIO0 în modul 0

În acest mod interfaţa este capabilă să transmită sau să recepţioneze mesaje de 8 biţi (mai întâi bitul mai puţin semnificativ), la o viteză de transmisie egală cu $f_{OSC}/12$.

Datele seriale sunt recepţionate sau transmise numai de pinul RXD, TXD fiind folosit pentru semnalul de ceas.

Transmisia este iniţiată de orice instrucţiune care scrie în registrul special S0BUF, activând semnalul SEND (emisie). Acest semnal setează pinul RXD să funcţioneze ca ieşire a registrului de deplasare şi, de asemenea, determină ca pinul TXD să funcţioneze ca ieşire a semnalului de ceas. Semnalul de ceas are o perioadă egală cu ciclul maşină, fiind 1 logic în stările S6, S1 şi S2, respectiv 0 logic în stările S3-S5. După transmiterea ultimului bit, semnalul SEND este dezactivat şi indicatorul TI este setat.

Recepţia este demarată de ştergerea bitului RI, ştergere condiţionată de setarea bitului REN. Registrul de control al recepţiei este încărcat cu valoarea 1 1111 1110 şi activează semnalul RECEIVE (recepţie). Cât timp acest semnal este activ, valoarea portului P3.0 (RXD) este încărcată şi deplasată la stânga o dată la fiecare ciclu maşină. În momentul în care valoarea zero, încărcată iniţial în poziţia celui mai puţin semnificativ bit, ajunge pe poziţia celui mai semnificativ bit se semnalează registrului de control al recepţiei să realizeze o ultimă deplasare şi încarcă registrul S0BUF. În următorul ciclu maşină semnalul RECEIVE este dezactivat şi indicatorul RI este setat.

Modul 0 este folosit, de regulă, pentru interfaţarea cu registre externe de deplasare TTL sau CMOS pentru extinderea numărului de porturi de intrare-ieşire.

1.9.2 Interfaţa serială SIO0 în modul 1

Procedura de lucru în acest mod permite transmiterea–recepţionarea a 10 biţi (un bit de start, 8 biţi date, un bit de stop) cu o viteză de transmisie variabilă, determinată de frecvenţa depăşirilor date de timerul T1.

Structura internă simplificată a interfeţei seriale în modul 1 este prezentată în figura 1.9.

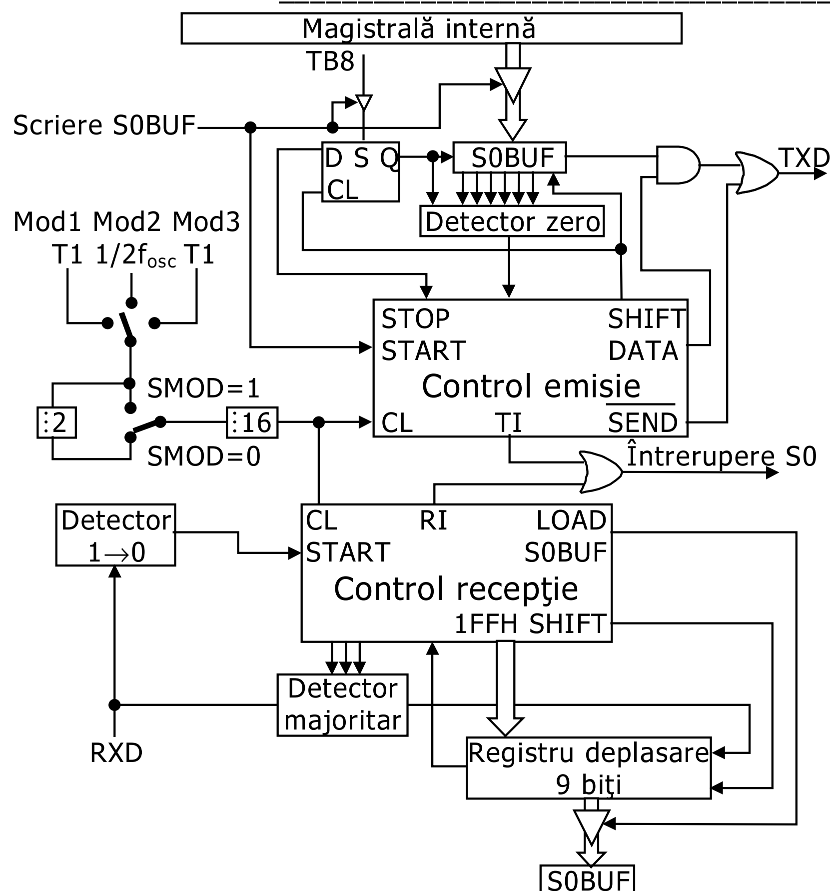


Figura 1.9. Interfaţa serială SIO0 (modurile 1, 2 şi 3)

Transmisia este iniţiată de orice instrucţiune care scrie în registrul special S0BUF. Adresarea S0BUF setează bitul 9 al registrului de serializare al transmisiei, înştiinţează unitatea de control că este solicitată o transmisie şi activează semnalul SEND (emisie). Transmisia începe în următorul ciclu maşină dar este sincronizată cu depăşirile date de timerul T1.

Semnalul SEND transmite un bit la ieşirea TXD (bitul de start), urmat de cei 8 biţi de date. În momentul în care al nouălea bit (bitul setat iniţial) din registrul de deplasare ajunge la extremitatea dreaptă a registrului, este semnalat unităţii de control a emisiei că mai este de emis un bit (bitul de stop), se dezactivează semnalul SEND şi este setat indicatorul TI.

Recepţia este declanşată de o tranziţie 1→0 a semnalului de pe pinul RXD. Pentru a asigura sincronizarea cu fluxul de date, iniţierea recepţiei produce o resetare imediată a timerului T1. De asemenea, registrul de serializare pentru recepţie este încărcat cu 1FFh. Deoarece tactul de deplasare provenit din timerul T1 este divizat cu 16, starea pinului RXD la impulsurile de tact cu numerele 7, 8 şi 9 determină ce valoare va avea bitul recepţionat: detectorul de biţi ia o decizie prin majoritate, valoarea acceptată fiind găsită în cel puţin două din cele trei stări 7, 8 şi 9. Dacă decizia majoritară în situaţia primului bit nu este 0, blocul de control deduce că a fost un fals impuls de start şi interfaţa este resetată.

Când bitul de start, deplasat succesiv, ajunge în registrul de deplasare pe poziţia limită stânga (al nouălea bit), se semnalează blocului de control al

recepţiei că a fost primit ultimul bit, încarcă S0BUF şi RB8 şi setează RI. Octetul recepţionat este disponibil dacă sunt îndeplinite fiecare din următoarele două condiţii: indicatorul RI şters şi SM2=0 sau bitul de stop=1.

1.9.3 Interfaţa serială SIO0 în modul 2

În modul 2 sunt emişi 11 biţi prin intermediul TXD sau recepţionaţi de RXD: un bit de start, 8 biţi de date, un bit programabil (bitul 9 – de regulă bit de paritate) şi un bit de stop. La emisie, bitul 9 poate fi programat prin S0CON.3 (TB8). La recepţie, bitul 9 este regăsit în S0CON.2 (RB8). Viteza de transmisie este selectabilă acceptând una din valorile $f_{OSC}/32$ sau $f_{OSC}/64$.

Structura internă este asemănătoare cu cea din figura 1.9; diferenţa este dată de existenţa unei surse suplimentare pentru generarea ratei de transmisie.

Transmisia este activată de scrierea registrului S0BUF. Scrierea în S0BUF încarcă TB8 în bitul 9 al registrului de deplasare şi semnalează blocului de control al transmisiei că este solicitată o emisie. Transmisia începe activarea semnalului SEND care emite bitul de start. După emiterea primului bit şi translatarea spre dreapta a conţinutului registrului de deplasare, pe poziţia bitului cel mai semnificativ este introdus 1. Ulterior, deplasarea celorlalţi biţi produce introducerea unor biţi 0, astfel că, în momentul în care TB8 a ajuns în poziţia de emisie din registrul de deplasare, este urmat de bitul de stop (setat pe 1), restul biţilor din registru fiind şterşi. Această condiţie este detectată de blocul de control al emisiei şi îi semnalează că mai are de emis un bit, după care dezactivează SEND şi setează indicatorul întreruperii la transmisie TI.

Recepţia este iniţiată de o tranziţie $1\rightarrow0$ a semnalului RXD. Pentru a asigura sincronizarea cu fluxul de date, iniţierea recepţiei produce o resetare imediată a timerului T1. De asemenea, registrul de serializare pentru recepţie este încărcat cu 1FFh.

Când bitul de start, deplasat succesiv, ajunge în registrul de deplasare pe poziţia limită stânga (al nouălea bit), se semnalează blocului de control al recepţiei că a fost primit ultimul bit, încarcă S0BUF şi RB8 şi setează RI. Octetul recepţionat este disponibil dacă sunt îndeplinite fiecare din următoarele două condiţii: indicatorul RI şters şi SM2=0 sau bitul de stop=1.

1.9.4 Interfaţa serială SIO0 în modul 3

În modul 3 sunt emişi 11 biţi prin intermediul TXD sau recepţionaţi de RXD: un bit de start, 8 biţi de date, un bit programabil (bitul 9 – de regulă bit de paritate) şi un bit de stop. La emisie, bitul 9 poate fi programat prin S0CON.3 (TB8). La recepţie, bitul 9 este regăsit în S0CON.2 (RB8). Viteza de transmisie este determinată de timerul T1.

Structura internă este asemănătoare cu cea din figura 1.9; diferenţa este dată de existenţa unei surse suplimentare pentru generarea vitezei de transmisie.

Cu excepţia ratei de transmisie, aici determinată de depăşirile timerului T1, funcţionarea interfeţei este identică cu modul 2.

Vitezele de transmisie utilizate curent sunt prezentate în tabelul 1.9.

Tabelul 1.9

Mod SIO0	Viteză transmisie	f_{OSC}	Timer 1			
			SMOD (PCON.7)	$C/\overline{T}$ (TMOD.6)	Mod	Valoare reîncărcare
0	1 MHz	12 MHz	X	X	X	X
2	375 kHz	12 MHz	1	X	X	X
	187,5 kHz	12 MHz	0	X	X	X
1, 3	62.5 kHz	12 MHz	1	0	2	FFh
	19.2 kHz	11.059 MHz	1	0	2	FDh
	9600 Hz	11.059 MHz	0	0	2	FDh
	4800 Hz	11.059 MHz	0	0	2	FAh
	2400 Hz	11.059 MHz	0	0	2	F4h
	1200 Hz	11.059 MHz	0	0	2	E8h
	137.5 Hz	11.059 MHz	0	0	2	1Dh
	110 Hz	6 MHz	0	0	2	72h
	110 Hz	12 MHz	0	0	1	FEEBh

Viteza de transmisie pentru modurile 1 şi 3 se poate calcula cu următoarea relaţie:

$$\text{rata transmisie} = \frac{\text{rată T1}}{(256 - \text{TH1}) \cdot (32 - \text{SMOD} \cdot 16)}$$

Setarea timerului T1 trebuie făcută funcţie de următoarele condiţii:

- TCON.6=1 pentru validarea funcţionării timerului T1;
- TMOD.5=1 şi TMOD.4=0 pentru timer T1 în mod 2 (timer de 8 biţi cu reîncărcare automată), respectiv TMOD.5=0 şi TMOD.4=1 pentru timer T1 în mod 1 (timer de 16 biţi – folosit pentru viteze mici de transmisie);
- TMOD.6=0 pentru a funcţiona cu divizarea frecvenţei oscilatorului intern sau TMOD.6=1 pentru a diviza o frecvenţă externă;
- IEN0.3=0 pentru dezactivarea întreruperilor timerului T1. Dacă T1 este în modul 1 (timer de 16 biţi) este necesară validarea întreruperii, scopul fiind reîncărcarea registrului TM2.

1.10. Interfaţa serială sincronă I^2C

Magistrala I^2C, realizată de Philips, permite schimbarea de date între unul sau mai multe dispozitive principale (master) şi un număr de dispozitive subordonate (slave), conectate pe două linii, SDA (date) şi SCL (ceas). Facilităţile principale ale magistralei I^2C sunt:

- transfer de date bidirecţional între master şi slave;
- arbitrarea coliziunilor de date;
- sincronizarea oferită de semnalul SCL permite comunicarea între periferice cu diferite viteze de transmisie;

- semnalul `SCL` poate fi folosit şi pentru controlul transmisiei între periferice, în sensul suspendării sau reluării transferului de date în orice moment.

Modulul serial sincron al familiei 8xC552 respectă specificaţiile magistralei I^{2}C, precum şi toate modurile de transfer (mai puţin modul de transfer cu viteză mică). Legăturile spre exterior ale interfeţei sincrone SIO1 sunt pinii `SCL` (`P1.6`) şi `SDA` (`P1.7`); pentru validarea interfeţei este obligatorie setarea celor doi pini ai portului `P1`.

O configuraţie tipică de magistrală I^{2}C este prezentată în figura 1.10.

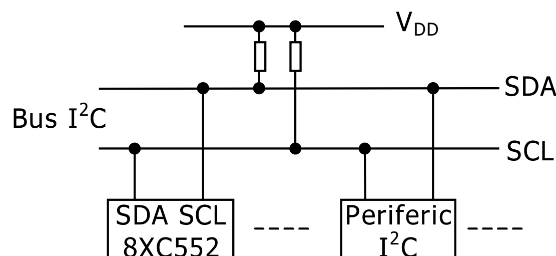


Figura 1.10. Configuraţia interfeţei I^{2}C

Interfaţa SIO1 poate lucra în patru moduri: emisie circuit principal, recepţie circuit principal, emisie circuit secundar sau recepţie circuit secundar.

Registrele speciale responsabile de funcţionarea interfeţei SIO1 sunt: `S1CON`, `S1ADR`, `S1DAT` şi `S1STA`. Structura lor este prezentată în tabelul 1.10.

Tabelul 1.10

S1CON (D8h)	CR2	ENSI	STA	STO	SI	AA	CR1	CR0

Bit	Descriere	CR2	CR1	CR0	f_{OSC}=6 MHz	f_{OSC}=12 MHz	Divizare f_{OSC}
CR2 CR1 CR0	Selectarea vitezei de transmisie	0	0	0	23 kHz	47 kHz	256
		0	0	1	27 kHz	54 kHz	224
		0	1	0	31 kHz	63 kHz	192
		0	1	1	37 kHz	75 kHz	160
		1	0	0	6.25 kHz	12.5 kHz	960
		1	0	1	50 kHz	100 kHz	120
		1	1	0	100 kHz	200 kHz	60
		1	1	1	(0.25…62.5)	(0.25…62.5)	96·(256-valoare T1)
ENSI	Dacă este şters, SDA şi SCL sunt în înaltă impedanţă şi orice semnale sunt ignorate.						
STA	Dacă bitul este şters nu va fi generată condiţia de START sau START repetat (START R). Dacă este setat, SIO1 care intră în mod master verifică magistrala I^{2}C şi dacă aceasta este liberă generează o condiţie START. Dacă SIO1 este deja în mod master, SIO1 transmite o condiţie START R. Bitul poate fi setat în orice moment, chiar dacă SIO1 este slave.						
STO	Indicator de stop. Dacă bitul este şters, nu va fi generată o condiţie STOP. Dacă bitul este setat şi SIO1 este în mod master, în momentul în care va fi detectat pe magistrală, circuitul va şterge indicatorul STOP. Dacă SIO1 este în mod slave, se va trimite numai o condiţie STOP internă. Oricum, dacă SIO1 se comportă ca şi cum ar fi recepţionată o condiţie STOP comută în regimul "inexistenţă adresă slave" şi indicatorul este şters automat.						
SI	Indicator întrerupere. Dacă este şters, nu vor fi generate întreruperi ale SIO1. Dacă este setat concomitent cu bitul EA şi ESI1 (din registrul IEN0), va fi generată o întrerupere în 25 din cele 26 de stări posibile ale SIO1. SI trebuie şters prin program.						
AA	Indicator confirmare. Dacă este şters, nu va fi returnată o stare validare pe timpul impulsului de confirmare a SCL. Dacă este setat, va fi returnată o stare de validare pe timpul impulsului de confirmare a SCL când s-a recepţionat: o adresă de slave, o adresă de apel general (bitul GC din S1ADR este setat) ori un octet de către SIO1 aflat în modul master sau slave.						

S1DAT (DAh)	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0

SD7…SD0	Octetul recepţionat sau transmis pe magistrala I^{2}C. S1DAT împreună cu indicatorul ACK formează un registru de deplasare de 9 biţi. Datele sunt deplasate pe frontul crescător al SCL şi trimise la pinul SDA pe frontul căzător al SCL prin intermediul unui buffer BSD7.							
S1ADR (DBh)	Adr6	Adr5	Adr4	Adr3	Adr2	Adr1	Adr0	GC
Adr6… Adr0	În mod master, conţinutul registrului este irelevant. În mod slave reprezintă adresa de identificare proprie a perifericului.							
GC	Dacă este setat, dispozitivul va recunoaşte starea de apel general.							
S1STA (D9h)	SC4	SC3	SC2	SC1	SC0	0	0	0
Cei cinci biţi pot genera 32 de stări, din care 26 posibile, funcţie de modul de lucru al interfeţei. Toate stările sunt prezentate în paragrafele 1.10.1-1.10.5.								

1.10.1 Modul emisie circuit principal

În acest mod, un număr de octeţi este transmis către un circuit secundar. Registrul S1CON trebuie setat în modul următor:

S1CON							
X	1	0	0	0	X	X	X

Modul emisie circuit principal poate fi acum iniţializat setând STA (S1CON.5). Logica modulului SIO1 va testa magistrala I^{2}C şi va genera un START când aceasta va fi liberă. După ce START a fost emis, se setează indicatorul întreruperii (SI) şi valoarea registrului de stare (S1STA) devine 08h. Codul de stare trebuie folosit pentru rutina de tratare a întreruperii care va încărca registrul S1DAT cu adresa dispozitivului secundar şi bitul de direcţie (SLA+W). SI trebuie şters prin program.

După ce acestea au fost emise şi bitul de confirmare de la circuitul secundar a fost recepţionat, este setat din nou SI şi S1STA poate conţine mai multe coduri descrise în tabelul 1.10.a.

Tabelul 1.10.a

Cod [H]	Stare I^{2}C	Răspuns program: S1DAT	STA	STO	SI	AA	Următoarea acţiune I^{2}C
08	S-a emis START.	Încarcă SLA+W	X	0	0	X	Va fi emis SLA+W. Se va recepţiona ACK.
10	S-a emis START R.	Încarcă SLA+W Încarcă SLA+R	X X	0 0	0 0	X X	Ca mai sus. Se emite SLA+R. SIO1 trece în modul b).
18	S-a emis SLA+W. ACK s-a recepţionat.	Încarcă octet Nici o acţiune S1DAT	0 1 0 1	0 0 1 1	0 0 0 0	X X X X	Octetul va fi transmis. Va fi transmis START R. Va fi transmis STOP. Va fi transmis STOP urmat de START.
20	S-a emis SLA+W. ACK nu s-a recepţionat.	Încarcă octet Nici o acţiune S1DAT	0 1 0 1	0 0 1 1	0 0 0 0	X X X X	Octetul va fi transmis. Va fi transmis START R. Va fi transmis STOP. Va fi transmis STOP urmat de START.
28	S-au emis datele. ACK s-a recepţionat.	Încarcă octet Nici o acţiune S1DAT	0 1 0 1	0 0 1 1	0 0 0 0	X X X X	Octetul va fi transmis. Va fi transmis START R. Va fi transmis STOP. Va fi transmis STOP urmat de START.
30	S-au emis datele. ACK nu s-a recepţionat.	Încarcă octet Nici o acţiune S1DAT	0 1 0 1	0 0 1 1	0 0 0 0	X X X X	Octetul va fi transmis. Va fi transmis START R. Va fi transmis STOP. Va fi transmis STOP urmat de START.

38	S-a pierdut arbitrarea SLA+R/W sau datele.	Nici o acţiune S1DAT	0 1	0 0	0 0	X X	Se eliberează magistrala. Se emite start când magistrala este liberă.

Protocolul legăturii în modul transmisie circuit principal este prezentat în figura 1.11.

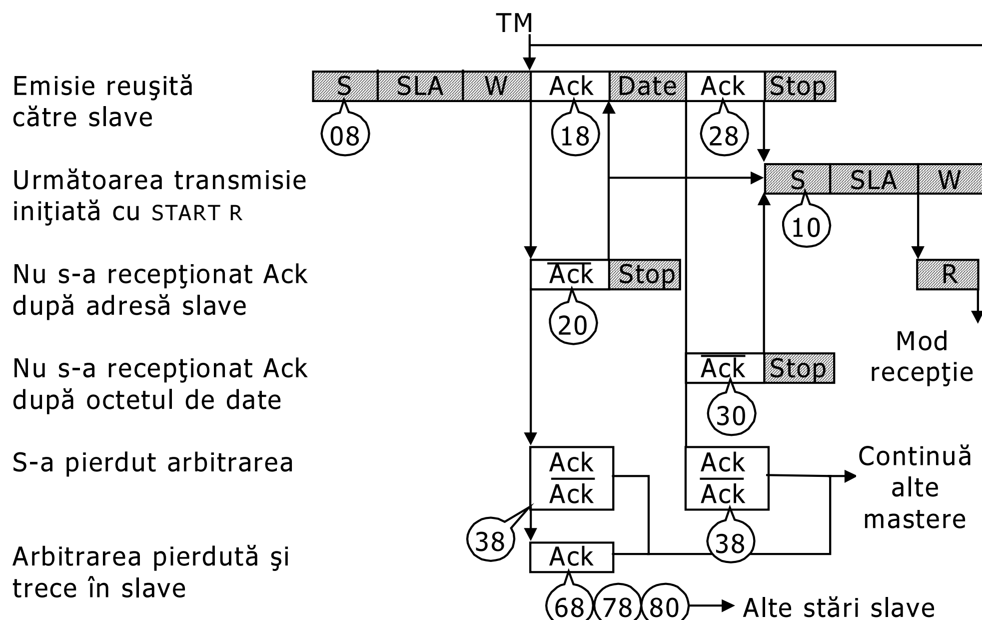


Figura 1.11. SIO1 în mod transmisie master (TM)

1.10.2 Modul recepţie circuit principal

În acest mod, un număr de octeţi este recepţionat de la un dispozitiv secundar. Transferul este iniţiat ca în modul transmisie circuit principal. După ce a fost emis START, rutina de tratare a întreruperii trebuie să încarce în S1DAT adresa dispozitivului secundar şi bitul de direcţie (SLA+R). Bitul SI trebuie şters de program. După emiterea SLA+R şi primirea de la circuitul secundar a confirmării, indicatorul SI este setat din nou şi registrul S1STA conţine o serie de coduri descrise în tabelul 1.10.b.

Tabelul 1.10.b

Cod	Stare I^2C	S1DAT	STA	STO	SI	AA	Următoarea acţiune I^2C
08	S-a emis START.	Încarcă SLA+R	X	0	0	X	Va fi emis SLA+R. Se va recepţiona ACK.
10	S-a emis START R.	Încarcă SLA+R Încarcă SLA+W	X X	0 0	0 0	X X	Ca mai sus. Va fi emis SLA+W. SIO1 comută în modul a).
38	S-a pierdut arbitrarea SLA+R sau ACK.	Nici o acţiune S1DAT	0 1	0 0	0 0	X X	Magistrala va fi eliberată. SIO1 va intra în modul slave neadresat. Va fi emis start dacă magistrala este liberă.
40	S-a emis SLA+W. ACK s-a recepţionat.	Nici o acţiune S1DAT	0 1	0 0	0 0	0 1	Octetul va fi recepţionat. Nu va fi emis ACK. Octetul va fi recepţionat. Va fi emis ACK.
48	S-a emis SLA+W. ACK nu s-a recepţionat.	Nici o acţiune S1DAT	1 0 1	0 1 1	0 0 0	X X X	Va fi emis START R. Va fi emis STOP. Va fi emis STOP urmat de START.

50	S-au recepţionat date. ACK a fost emis.	Citeşte octet	0	0	0	0	Octetul va fi recepţionat. Nu va fi emis ACK.
			0	0	0	1	Octetul va fi recepţionat. Va fi emis ACK.
58	S-au recepţionat datele. ACK nu a fost emis.	Citeşte octet	1	0	0	X	Va fi emis START R.
			0	1	0	X	Va fi emis STOP.
			1	1	0	X	Va fi emis STOP urmat de START.

Protocolul legăturii în modul recepţie circuit principal este prezentat în figura 1.12.

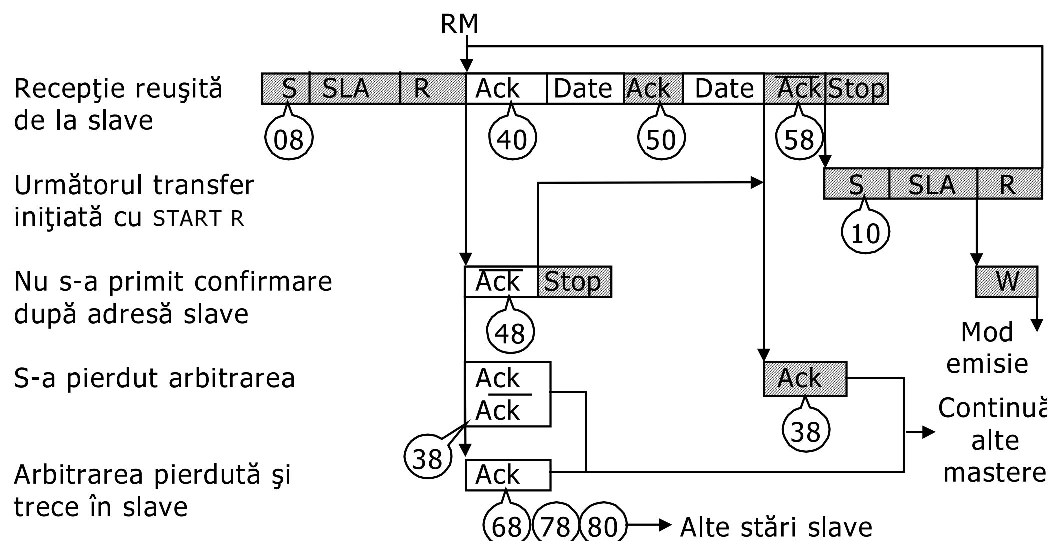


Figura 1.12. SIO1 în mod recepţie circuit principal

1.10.3 Modul recepţie circuit secundar

În acest mod, registrul S1ADR al dispozitivului secundar trebuie încărcat cu adresa cu care acesta este identificat de către master. Bitul GC (S1ADR.0) este setat dacă se doreşte ca dispozitivul să răspundă la apelul general. Registrul S1CON trebuie setat în modul următor:

S1CON								
	X	1	0	0	0	1	X	X

După încărcarea celor două registre, SIO1 intră în aşteptare până când este apelat prin adresa proprie şi bitul de direcţie (în acest caz W=0). După recepţionarea SLA+W este setat SI, iar în S1STA este găsit un cod folosit de rutina de tratare a întreruperii (tabelul 1.10.c). Acest mod mai poate fi activ în urma pierderii arbitrării când SIO1 era în mod circuit principal (stările 68h şi 78h).

Tabelul 1.10.c

Cod	Stare I^2C	S1DAT	STA	STO	SI	AA	Următoarea acţiune I^2C
60	S-a recepţionat SLA+W propriu. ACK a fost emis.	Nici o acţiune S1DAT	X	0	0	0	Octetul va fi recepţionat. Nu va fi emis ACK.
			X	0	0	1	Octetul va fi recepţionat. Va fi emis ACK.
68	S-a pierdut arbitrarea. S-a recepţionat SLA+W propriu. ACK a fost emis.	Nici o acţiune S1DAT	X	0	0	0	Octetul va fi recepţionat. Nu va fi emis ACK.
			X	0	0	1	Octetul va fi recepţionat. Va fi emis ACK.
70	S-a recepţionat un apel general. ACK a fost emis.	Nici o acţiune S1DAT	X	0	0	0	Octetul va fi recepţionat. Nu va fi emis ACK.
			X	0	0	1	Octetul va fi recepţionat. Va fi emis ACK.
78	S-a pierdut arbitra-	Nici o acţiune	X	0	0	0	Octetul va fi recepţionat. Nu va fi emis

	rea. S-a recepţionat un apel general. ACK a fost emis.	S1DAT	X	0	0	1	ACK. Octetul va fi recepţionat. Va fi emis ACK.
80	Adresat anterior cu adresa proprie. S-au recepţionat datele. ACK a fost emis.	Citeşte datele	X X	0 0	0 0	0 1	Octetul va fi recepţionat. Nu va fi emis ACK. Octetul va fi recepţionat. Va fi emis ACK.
88	Adresat anterior cu adresa proprie. S-au recepţionat datele. ACK nu a fost emis.	Citeşte datele	0 0 1 1	0 0 0 0	0 0 0 0	0 1 0 1	Comutare în modul slave neadresat. Nu este recunoscută adresa proprie slave sau de apel general. Comutare în modul slave neadresat. Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Comutare în modul slave neadresat. Nu este recunoscută adresa proprie slave sau de apel general. Se emite start când magistrala este liberă. Comutare în modul slave neadresat. Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Se emite start când magistrala este liberă.
90	Adresat anterior cu adresa apel general. S-au recepţionat datele. ACK a fost emis.	Citeşte datele	X X	0 0	0 0	0 1	Octetul va fi recepţionat. Nu va fi emis ACK. Octetul va fi recepţionat. Va fi emis ACK.
98	Adresat anterior cu adresa apel general. S-au recepţionat datele. ACK nu a fost emis.	Citeşte datele	0 0 1 1	0 0 0 0	0 0 0 0	0 1 0 1	Nu este recunoscută adresa proprie slave sau de apel general. Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Nu este recunoscută adresa proprie slave sau de apel general. Se emite START când magistrala este liberă. Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Se emite START când magistrala este liberă.
A0	S-a recepţionat STOP sau START R când era adresat în modurile c) sau d)	Citeşte datele	0 0 1 1	0 0 0 0	0 0 0 0	0 1 0 1	Nu este recunoscută adresa proprie slave sau de apel general. Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Nu este recunoscută adresa proprie slave sau de apel general. Se emite START când magistrala este liberă. Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Se emite START când magistrala este liberă.

Dacă bitul AA (S1CON.2) este şters se va emite un bit de non-confirmare după următorul octet primit. Cât timp AA este şters, SIO1 nu va răspunde la adresa proprie sau la apelul general dar dispozitivul verifică în continuare linia şi recunoaşterea adresei poate fi refăcută prin setarea bitului AA.

Protocolul legăturii în modul recepţie circuit secundar este prezentat în figura 1.13.

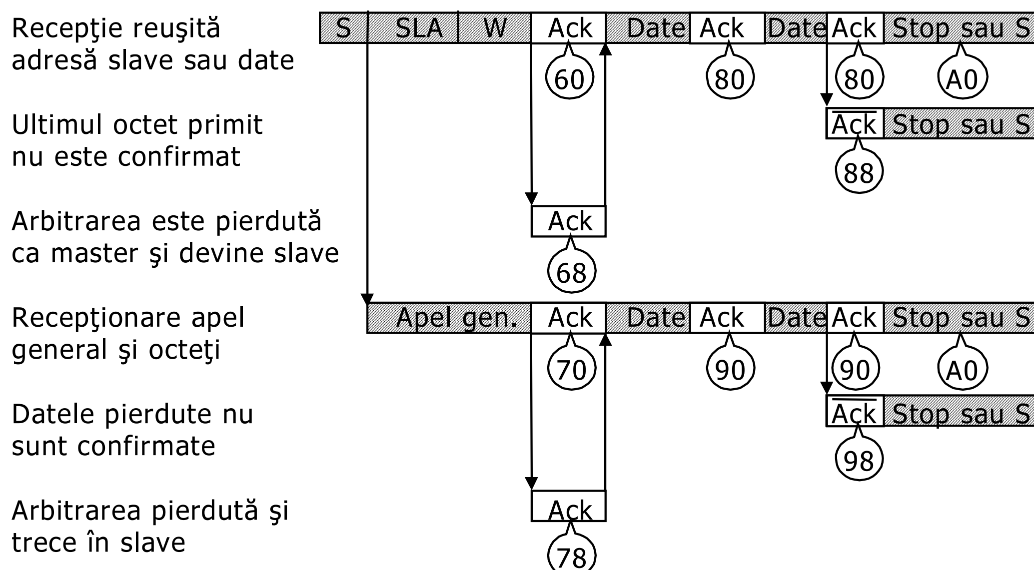


Figura 1.13. SIO1 în mod recepţie circuit secundar

1.10.4 Modul transmisie circuit secundar

În acest mod, un număr de octeţi este transmis către un dispozitiv principal receptor. Transferul datelor este iniţializat ca în modul c). După scrierea S1ADR şi S1DAT, SIO1 aşteaptă până este apelat prin adresa proprie şi bitul de direcţie (în această situaţie R=1). După ce SLA+R a fost citit, indicatorul SI este setat şi în S1STA este găsit un cod folosit de rutina de tratare a întreruperii (tabelul 1.10.d.). Acest mod mai poate fi activ în urma pierderii arbitrării când SIO1 era în mod circuit principal (starea B0h).

Dacă bitul AA (S1CON.2) este şters, SIO1 va transmite ultimul octet şi va intra în starea C0h sau C8h. Modulul va comuta în modul slave neadresat şi va ignora dispozitivul master care va recepţiona numai biţi de 1 logic. Cât timp AA este şters, SIO1 nu va răspunde la adresa proprie sau la apelul general, dar dispozitivul verifică în continuare linia şi recunoaşterea adresei poate fi refăcută prin setarea bitului AA.

Tabelul 1.10.d

Cod	Stare I^2C	S1DAT	STA	STO	SI	AA	Următoarea acţiune I^2C
A8	S-a recepţionat SLA+W propriu. ACK a fost emis.	Încarcă data	X	0	0	0	Ultimul octet va fi emis. ACK va fi recepţionat.
			X	0	0	1	Octetul va fi emis. ACK va fi recepţionat.
B0	S-a pierdut arbitrarea SLA+R/W. S-a recepţionat SLA+R propriu. ACK a fost emis.	Încarcă data	X	0	0	0	Ultimul octet va fi emis. ACK va fi recepţionat.
			X	0	0	1	Octetul va fi emis. ACK va fi recepţionat.
B8	Octetul din S1DAT a fost emis. a ACK a fost primit.	Încarcă data	X	0	0	0	Ultimul octet va fi emis. ACK va fi recepţionat.
			X	0	0	1	Octetul va fi emis. ACK va fi recepţionat

C0	Octetul din S1DAT a fost emis. ACK nu a fost primit.	Nici o acţiune S1DAT	0	0	0	0	Nu este recunoscută adresa proprie slave sau de apel general.
			0	0	0	1	Este recunoscută adresa proprie slave şi de apel general dacă GC=1.
			1	0	0	0	Nu este recunoscută adresa proprie slave sau de apel general. Se emite START când magistrala este liberă.
			1	0	0	1	Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Se emite START când magistrala este liberă.
C8	Ultimul octet din S1DAT a fost emis. ACK a fost primit.	Nici o acţiune S1DAT	0	0	0	0	Nu este recunoscută adresa proprie slave sau de apel general.
			0	0	0	1	Este recunoscută adresa proprie slave şi de apel general dacă GC=1.
			1	0	0	0	Nu este recunoscută adresa proprie slave sau de apel general. Se emite START când magistrala este liberă.
			1	0	0	1	Este recunoscută adresa proprie slave şi de apel general dacă GC=1. Se emite START când magistrala este liberă.

Protocolul legăturii în modul emisie circuit secundar este prezentat în figura 1.14.

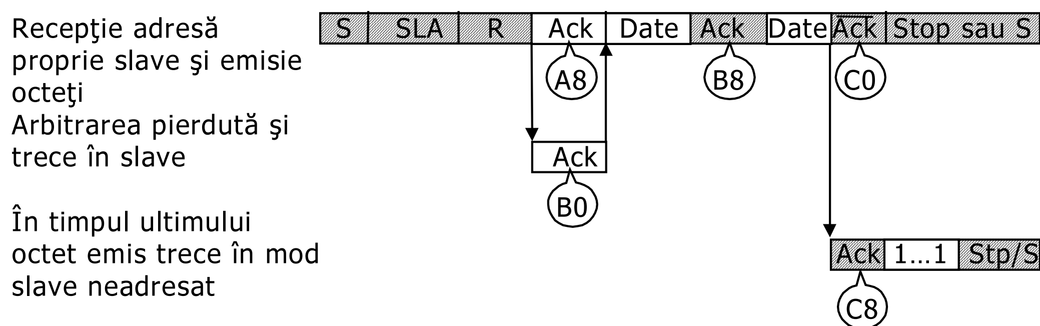


Figura 1.14. SIO1 în mod transmisie circuit secundar

1.10.5 Alte stări

Cu excepţia stărilor a)...d), registrul S1STA mai poate defini două ipostaze ale interfeţei I^2C, prezentate în tabelul 1.10.e.

Tabelul 1.10.e

Cod	Stare I^2C	S1DAT	STA	STO	SI	AA	Următoarea acţiune I^2C
F8	Nici o informaţie pertinentă. SI=0.	Nici o acţiune S1DAT şi S1CON					SIO1 este în aşteptare sau realizează un transfer.
00	Eroare magistrală	Nici o acţiune S1DAT	0	1	0	X	În toate cazurile magistrala este eliberată şi SIO1 trece în modul slave neadresat.

1.11. Sistemul de întreruperi

Familia 8XC552 are 15 întreruperi, fiecare putând fi asignată la unul din cele două nivele de prioritate.

Întreruperile externe $\overline{INT0}$ şi $\overline{INT1}$ pot fi programate să fie active pe front sau pe nivel funcţie de biţii IT0 şi IT1 din registrul special TCON. Indicatorii acestor întreruperi sunt IE0 şi IE1 din TCON.

Întreruperile timerelor T0 şi T1 sunt controlate de indicatorii TF0 şi TF1 care sunt setate de depăşirea registrelor de numărare (excepţie o face T1 în modul 3 – mai multe informaţii se găsesc în paragraful 1.7.1.).

Cele opt întreruperi ale timerului T2 sunt generate de indicatorii CTI0-CTI3 (setate de semnalele de intrare CT0I-CT3I), CMI0-CMI2 (setat de egalitatea între registrul T2 şi registrele CM0-CM2) şi un SAU LOGIC între biţii T2BO şi T2OV (setaţi de depăşirea registrului de 8 biţi, respectiv 16 biţi).

Întreruperea convertorului analog/numeric este generată de indicatorul ADCI care este setat în momentul în care rezultatul conversiei este gata de citit. Întreruperea interfeţei asincrone (SIO0) este generată de o operaţie SAU LOGIC între indicatorii TI (transmisie) şi RI (recepţie). Întreruperea interfeţei sincrone (SIO1) este generată de indicatorul SI setat de o stare validă în registrul S1STA.

Sistemul de întreruperi este controlat de un set de registre de validare (IEN0 şi IEN1) şi registrele de prioritate (IP0 şi IP1). Structura acestora este prezentată în tabelul 1.11.

Indicatorii întreruperilor sunt verificaţi în fiecare ciclu maşină în starea S5P2. Dacă un indicator este setat, sistemul de întreruperi va genera o instrucţiune LCALL către rutina de tratare în ciclul maşină următor detectării indicatorului setat.

În anumite condiţii, instrucţiunea LCALL poate fi blocată:

- o întrerupere cu prioritate egală sau superioară este în lucru;
- ciclul maşină curent nu este ultimul ciclu al unei instrucţiuni;
- instrucţiunea curentă este o instrucţiune RETI sau de scriere-citire în unul din registrele IP0, IP1, IE0 sau IE1.

Tabelul 1.11

IEN0 (A8h)	EA	EAD	ES1	ES0	ET1	EX1	ET0	EX0
EA	Control global al întreruperilor. Dacă este şters, toate întreruperile sunt dezactivate.							
EAD	Validare întrerupere convertor analog/numeric.							
ES1	Validare întrerupere interfaţă serială SIO1.							
ES0	Validare întrerupere interfaţă serială SIO0.							
ET1	Validare întrerupere depăşire timer T1.							
EX1	Validare întrerupere externă 1.							
ET0	Validare întrerupere depăşire timer T0.							
EX0	Validare întrerupere externă 0.							
IEN1 (E8h)	ET2	ECM2	ECM1	ECM0	ECT3	ECT2	ECT1	ECT0
ET2	Validare întrerupere depăşire timer T2.							
ECM2	Validare întrerupere comparator 2.							
ECM1	Validare întrerupere comparator 1.							
ECM0	Validare întrerupere comparator 0.							
ECT3	Validare întrerupere captură 3.							
ECT2	Validare întrerupere captură 2.							
ECT1	Validare întrerupere captură 1.							
ECT0	Validare întrerupere captură 0.							

IP0 (B8h)	–	PAD	PS1	PS0	PT1	PX1	PT0	PX0

PAD	Prioritate întrerupere convertor analog/numeric.
PS1	Prioritate întrerupere interfaţă serială SIO1.
PS0	Prioritate întrerupere interfaţă serială SIO0.
PT1	Prioritate întrerupere depăşire timer T1.
PX1	Prioritate întrerupere externă 1.
PT0	Prioritate întrerupere depăşire timer T0.
PX0	Prioritate întrerupere externă 0.

IEN1 (E8h)	PT2	PCM2	PCM1	PCM0	PCT3	PCT2	PCT1	PCT0

PT2	Prioritate întrerupere depăşire timer T2.
PCM2	Prioritate întrerupere comparator 2.
PCM1	Prioritate întrerupere comparator 1.
PCM0	Prioritate întrerupere comparator 0.
PCT3	Prioritate întrerupere captură 3.
PCT2	Prioritate întrerupere captură 2.
PCT1	Prioritate întrerupere captură 1.
PCT0	Prioritate întrerupere captură 0.

Trebuie subliniat faptul că dacă un indicator de întrerupere este activ dar rutina nu este activată datorită condiţiilor de mai sus, există posibilitatea ca întreruperea să fie pierdută dacă după dispariţia condiţiilor de blocare indicatorul de întrerupere este şters.

LCALL introduce în stivă valoarea curentă a contorului program şi îl încarcă cu adresa de tratare a întreruperii, conform tabelului 1.12.

Tabelul 1.12

Sursă întrerupere	Adresă	Sursă întrerupere	Adresă	Sursă întrerupere	Adresă
Întrerupere ext. 0	0003h	Interfaţă SIO1	002Bh	Convertor A/N	0053h
Timer T0	000Bh	Captură 0	0033h	Comparare 0	005Bh
Întrerupere ext. 1	0013h	Captură 1	003Bh	Comparare 1	0063h
Timer T1	001Bh	Captură 2	0043h	Comparare 2	006Bh
Interfaţă SIO0	0023h	Captură 3	004Bh	Timer T2	0073h

Întoarcerea în programul principal este asigurată de instrucţiunea RETI care restaurează conţinutul contorului program cu valoarea dinainte de întrerupere.

1.12. Consumul redus de energie

Consumul redus de energie este o facilitate deosebită a familiei 8xC552 fiind extrem de utilă în situaţia controlului unor aparate portabile alimentate la baterii. Funcţie de structura circuitelor externe, introducerea controlerului într-un mod economic de lucru poate asigura reducerea consumului de energie cu câteva ordine de mărime.

Familia 8xC552 are două regimuri de lucru cu consum redus de energie:

a) Inactiv (IDLE);
b) Oprit (POWER-DOWN).

Registrul special care controlează, printre altele, regimul economic de funcţionare este PCON, descris în tabelul 1.13.

Tabelul 1.13

PCON (87h)	SMOD	–	–	WLE	GF1	GF0	PD	IDL

SMOD	Folosit de SIO0. Dacă este setat dublează viteza de transmisie în modurile 1, 2 şi 3.
WLE	Validare timer T3.
GF1, GF0	Indicator de uz general
PD	Dacă este setat, controlerul intră în modul de lucru POWER-DOWN. Este condiţionat de semnalul extern $\overline{EW}$.
IDL	Dacă este setat, controlerul intră în modul de lucru IDLE.

1.12.1 Modul inactiv

În acest mod rămân active: timerele T0, T1 şi T3, interfeţele externe SIO0 şi SIO1, precum şi întreruperile externe 0 şi 1.

Registrele stivă, acumulator, contor program, celelalte registre interne şi memoria RAM îşi păstrează conţinutul.

Ieşirea din acest mod este posibilă prin două metode:

- orice întrerupere provoacă ştergerea bitului IDL (PCON.0) ieşind astfel din acest mod. Întreruperea va fi servită iar următoarea instrucţiune după RETI va readuce contorul program la valoarea iniţială înaintea intrării în modul economic.
- modul inactiv poate fi dezactivat printr-un reset.

1.12.2 Modul oprit

În acest mod, oscilatorul circuitului este oprit şi toate modulele interne sunt blocate. Memoria RAM internă, inclusiv registrele speciale, îşi păstrează valoarea dinainte. Pentru o reducere de consum mai importantă, este posibilă şi reducerea tensiunii de alimentare până la o valoare la care memoria RAM îşi mai menţine conţinutul.

Ieşirea din acest mod se poate face numai printr-o iniţializare externă.

Familia de microcontrolere 80C16x

Circuitele 80C16x constituie una din cele mai reprezentative familii de controlere de 16 biţi. Ele combină performanţele extrem de ridicate ale unităţii centrale (până la 20 milioane de instrucţiuni pe secundă) cu o gamă largă de periferice foarte utile.

Familia 16x, nefiind constrânsă de necesitatea păstrării compatibilităţii cu alte familii anterioare, a putut fi realizată la parametri deosebiţi, principalele facilităţi fiind:

- unitate centrală de 16 biţi, cu frecvenţe de ceas de 20 MHz sau 40 MHz;
- majoritatea instrucţiunilor sunt executate într-un ciclu maşină, în special datorită adoptării unei arhitecturi cu stivă de instrucţiuni cu patru nivele:
  - timp de execuţie a majorităţii instrucţiunilor – 50 ns ;
  - timpi de execuţie pentru înmulţire (16×16 biţi) – 250 ns, împărţire (32/16 biţi) – 500 ns;
- spaţiu liniar de adresare pentru memoria program şi memoria de date este 16 MB;
- conţine o memorie internă: RAM–2 kB (din care 1024 destinate registrelor speciale) şi ROM–8 kB;
- caracteristicile magistralei externe sunt programabile pentru diferite structuri ale sistemului:
  - magistrală externă de date pe 8 biţi sau 16 biţi;
  - magistrală externă adrese-date multiplexată sau nemultiplexată;
  - dispune de semnale pentru arbitrarea magistralei externe
- pot fi programate 5 semnale speciale pentru selectare circuitelor externe;
- convertor A/D de 10 biţi cu 16 intrări şi 9.7 ms timp de conversie;
- două module multifuncţionale cu 5 timere;
- două module de captură-comparare cu 16 canale;
- timer programabil pentru resetare (watchdog);
- patru module generatoare de impulsuri modulate în durată;
- două interfeţe seriale (una de viteză mică, sincronă-asincronă, cealaltă de viteză mare, numai sincronă);
- până la 111 linii de intrare-ieşire care se pot configura separat ca intrare (standard ori trigger Schmitt) sau ieşire (push-pull ori drenă în gol);
- un sistem de întreruperi special realizat pentru sisteme în timp real (16 nivele de întrerupere cu 56 de întreruperi cu vectori separaţi, timpul mediu de răspuns la întrerupere fiind de 300-500 ns);
- conţine o structură asemănătoare DMA denumită PEC (*Peripheal Event Controler* – Controler pentru evenimente de la periferice) utilă pentru transferul unor blocuri de date într-un timp foarte scurt;
- are implementate soft două moduri de lucru pentru economisirea energiei (inactiv şi oprit).

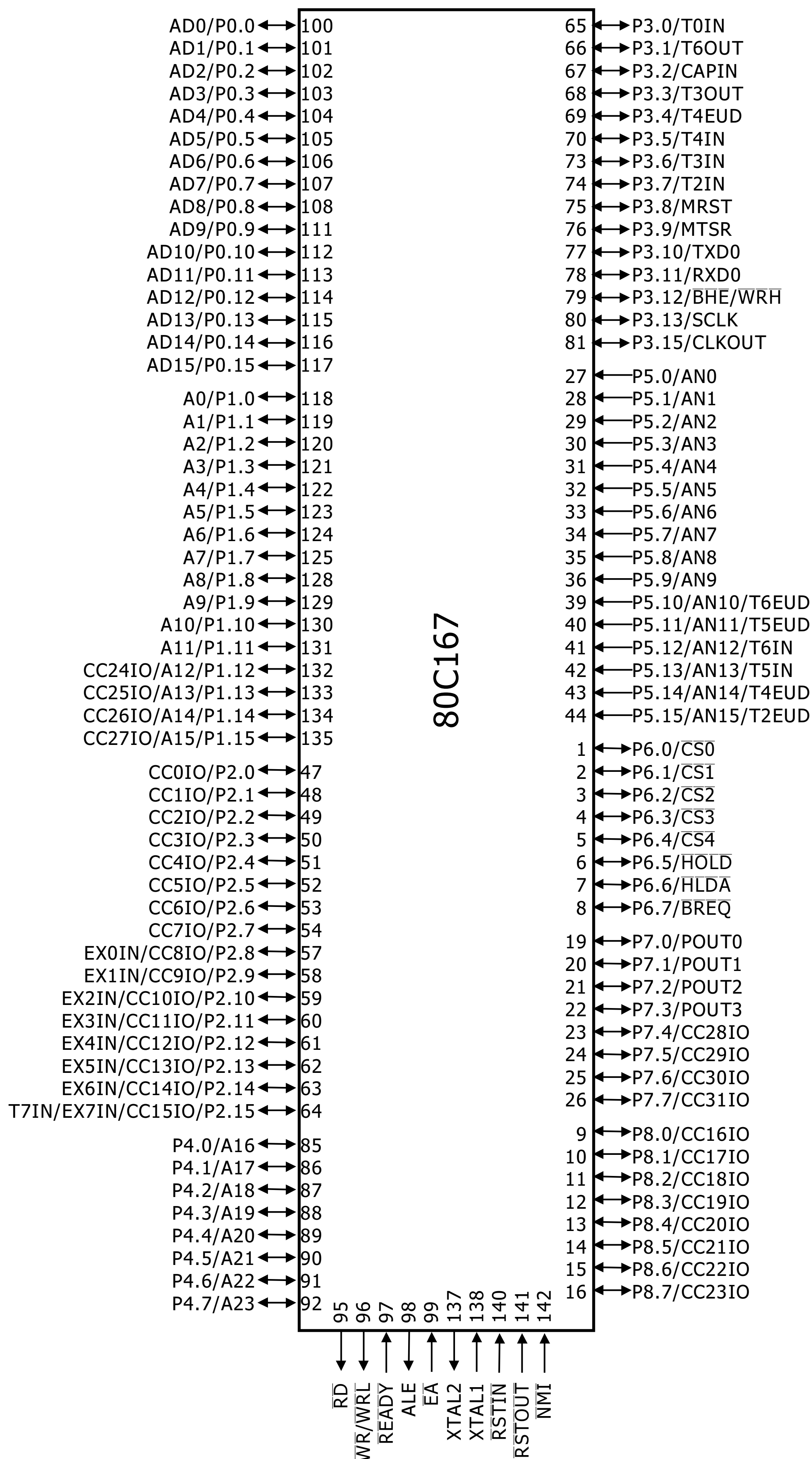


Figura 1.15. Microcontrolerul 80C167

Funcţie de tipul structurii interne, cei trei membri ai familiei 80C16x sunt:

- 80C166, controler de 16 biţi din prima generaţie, fără magistrală X-BUS;
- 80C165, controler din generaţia a doua, nu conţine modulele pentru convertorul analog/numeric, modulatoarele de impulsuri în durată şi registrele de captură şi comparare asociate timerelor;
- 80C167, varianta maximă de echipare, în structura sa internă fiind prezente toate modulele standard.

La generaţia a doua, datorită prezenţei magistralei X-BUS se pot realiza, la comandă, circuite specifice care pot conţine module suplimentare: memorie PROM sau Flash ROM, CAN (standard de comunicaţie serială creat de firma Bosch; termenul reprezintă *Controller Area Network* – reţea locală de controlere) sau alte circuite.

Descrierea funcţională a pinilor circuitului 80C167, cel mai reprezentativ membru al familiei, este prezentată în figura 2.1, iar structura internă în figura 2.2.

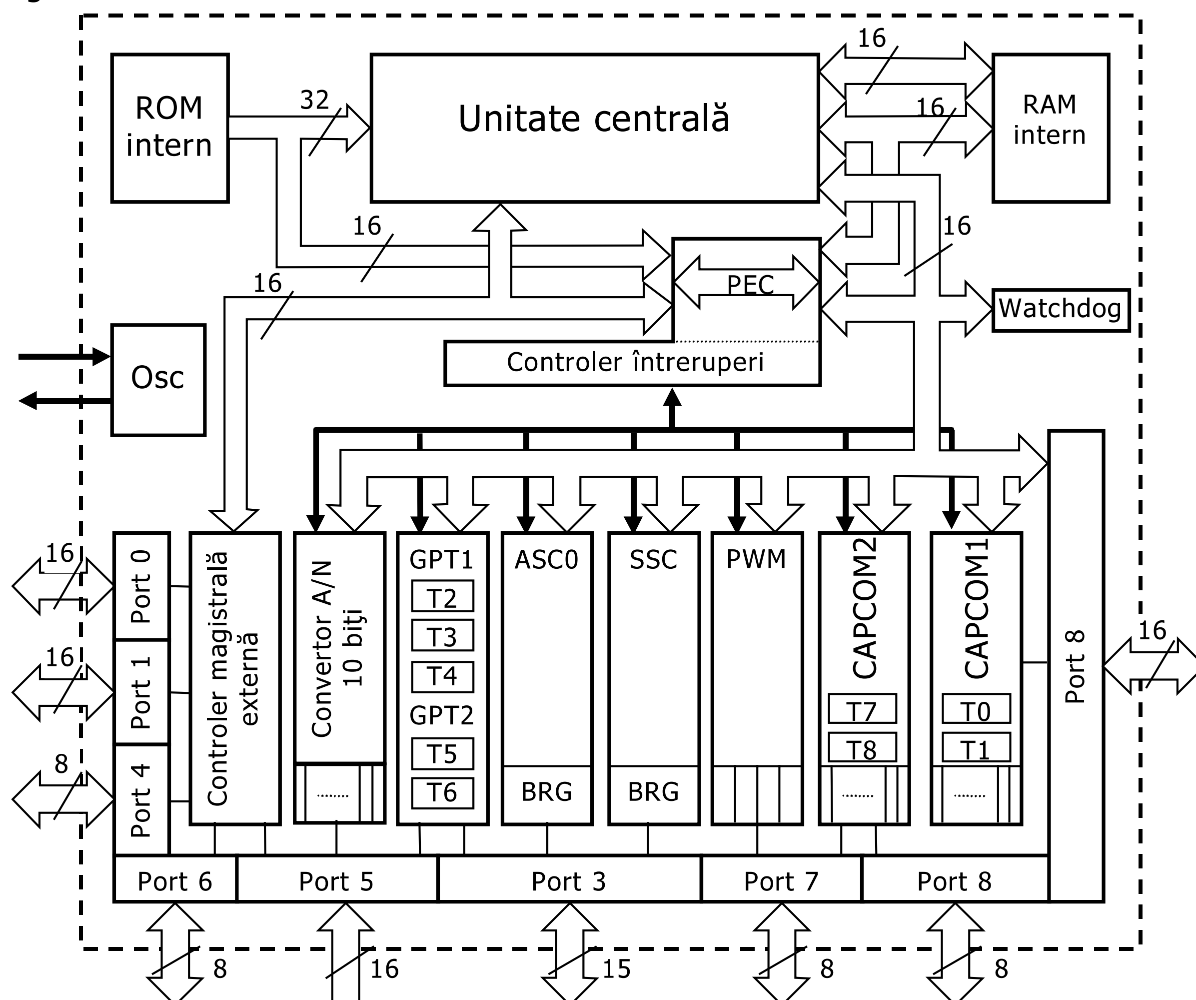


Figura 1.16. Structura internă a circuitului 80C167

Nucleul de bază al controlerului constă într-o unitate centrală care conţine o unitate aritmetică şi logică de 16 biţi, o stivă (*pipeline*) de 4 instrucţiuni, o unitate aritmetică separată pentru înmulţire şi împărţire, un registru de deplasare şi un generator pentru mascare la nivel de bit.

Legătura între unitatea centrală şi exterior, inclusiv celelalte module integrate în circuit este realizată de un controler special, interfaţa magistralei externe, o magistrală de 16 biţi. Controlul legăturii cu circuitele externe este complet la dispoziţia utilizatorului: se pot selecta patru mărimi ai magistrale de adrese (16, 18, 20 sau 24 de biţi), două tipuri de magistrale de date (8 biţi sau 16 biţi), magistralele de date putând fi multiplexate sau nu; de asemenea, se pot genera un număr de până la 5 semnale de selecţie de circuite, se poate programa poziţia şi lungimea semnalelor ALE şi $\overline{RW}$, se pot introduce automat 0…15 stări de aşteptare sau, pentru periferice foarte lente, semnale speciale de întârziere.

Circuitul mai conţine şi un sistem programabil de întreruperi cu priorităţi multiple care gestionează 56 de evenimente externe, produse de modulele interne, externe sau de program.

Informaţii suplimentare pot fi găsite în lucrările: *C167 16-Bit CMOS Single-Chip Microcontroller Data Sheet*, *C167 16-Bit Single-Chip Microcontroler User's Manual*, şi la adresa http://www.infineon.com/products/micro/.

1.13. Organizarea memoriei

Spaţiul de memorie al familiei de circuite 80C16x este configurat într-o arhitectură Von Neumann, adică instrucţiunile (codurile) şi datele sunt accesate în acelaşi spaţiu liniar de adresare. Toate zonele de memorie, separate fizic, incluzând memoriile ROM şi RAM interne, zona registrelor speciale (SFR) şi zona extinsă a registrelor speciale (ESFR), regiunea adreselor pentru perifericele XBUS, precum şi memoria externă sunt organizate în acelaşi spaţiu comun de adrese.

Circuitul 80C167 dispune de un spaţiu total de adresare de 16 MB. Acest spaţiu este aranjat în 256 segmente de 64 kB, fiecare segment la rândul lui fiind împărţit la rândul lui în patru pagini de date a câte 16 kB. Schematic, organizarea memoriei circuitului 80C167 este prezentată în figura 2.3.

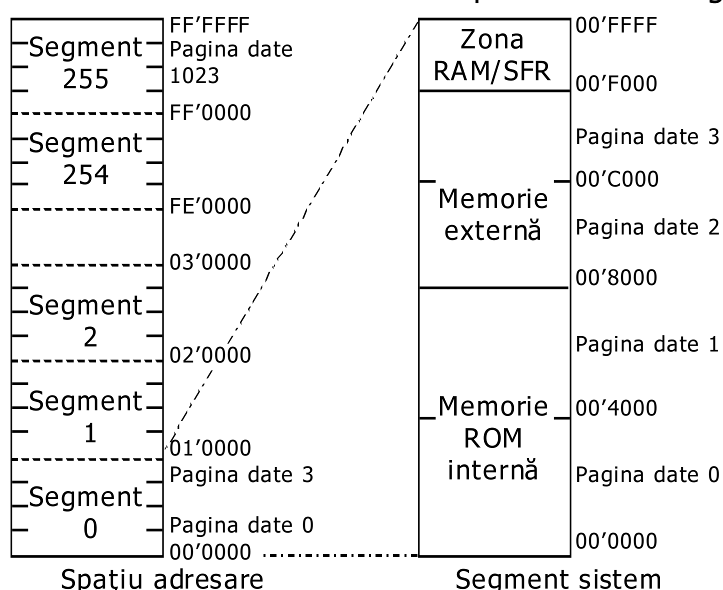


Figura 1.17. Harta memoriei circuitului 80C167

Cea mai mare parte a memoriei interne este apelată în segmentul 0, segmentul sistem. Partea superioară a segmentului sistem (4 kB – 00’F000h…00’FFFFh) conţin memoria RAM internă şi registrele speciale (SFR şi ESFR). Zona inferioară a segmentului 0 (32 kB – 00’0000h…00’7FFFh) poate fi ocupată de memoria ROM internă. Funcţie de conţinutul registrului special SYSCON, memoria ROM internă poate fi adresată din segmentul 1 (01’0000h…01’7FFFh) pentru a permite accesul memoriei externe şi în jumătatea inferioară a segmentului sistem.

Datele şi codurile pot fi memorate în orice zonă a memoriei interne, cu excepţia blocurilor rezervate pentru registrele speciale.

Octeţii (8 biţi) pot fi memoraţi la adrese pare sau impare. Cuvintele (16 biţi) sunt memorate la locaţii în ordine crescătoare, octetul inferior la adresă pară urmat de octetul superior memorat la o adresă impară. Cuvintele duble (32 biţi – numai coduri instrucţiuni) sunt memorate în locaţii succesive de memorie ca două cuvinte subsecvente. Biţii sunt memoraţi întotdeauna în poziţia specificată a bitului din cuvântul adresat; bitul 0 corespunde bitului cel mai puţin semnificativ al octetului, în timp ce bitul 15 corespunde bitului cel mai semnificativ. Adresarea pe bit este suportată de o parte a registrelor speciale şi a memoriei RAM interne şi, în totalitate, de registrele de uz general (GPR).

1.13.1 Memoria ROM internă

Circuitul 80C167 poate rezerva o zonă de 32 kB pentru o memorie ROM sau flash organizată ca 32×. Memoria ROM internă este validată sau invalidată global prin intermediul registrului special SYSCON, funcţie de starea pinului $\overline{\mathrm{EA}}$ la reset sau de comenzi ulterioare. Memoria ROM internă poate fi folosită atât pentru instrucţiuni, cât şi pentru date (constante, tabele de conversie etc.). Încărcarea codurilor este făcută întotdeauna de la adrese pare. Accesul datelor, octeţi sau cuvinte, este făcut prin intermediul adresării indirecte sau directe pe 16 biţi; pentru memoria ROM internă nu există posibilitatea adresării pe 8 biţi. Accesul memoriei interne, pentru dispozitivele care nu au incluse acest bloc, produce rezultate impredictibile.

1.13.2 Memoria RAM internă şi zona registrelor speciale (SFR)

Zona RAM/SFR se găseşte în pagina de date 3 şi permite accesul la 2 kB de memorie RAM (organizată ca 1k×16) şi la două blocuri a câte 512 octeţi de registre speciale.

De regulă, memoria RAM este utilizată pentru:

- stivă sistem (cu o mărime programabilă);
- bancuri de registre de uz general (GPR – *General Purpose Register*);
- indicatori sursă şi destinaţie pentru interfaţa pentru evenimente de la periferice (PEC – *Peripheal Event Controller*);
- memorare variabile şi alte date;
- memorare instrucţiuni.

Organizarea memoriei RAM interne este prezentată în figura 2.4. (zona haşurată reprezintă locaţii adresabile la nivel de bit).

Orice cuvânt sau octet din memoria RAM internă poate fi adresat indirect sau direct pe 16 biţi dacă indicatorul paginii de date utilizat (DPPx) este setat pentru pagina 3 de memorie. Accesul cuvintelor se face de la adrese pare. Pentru transferul datelor prin intermediul interfeţei pentru evenimente de la periferice (PEC), memoria RAM internă este accesată indiferent de conţinutul registrelor DPP, contând numai indicatorii sursă şi destinaţie corespunzători canalului PEC.

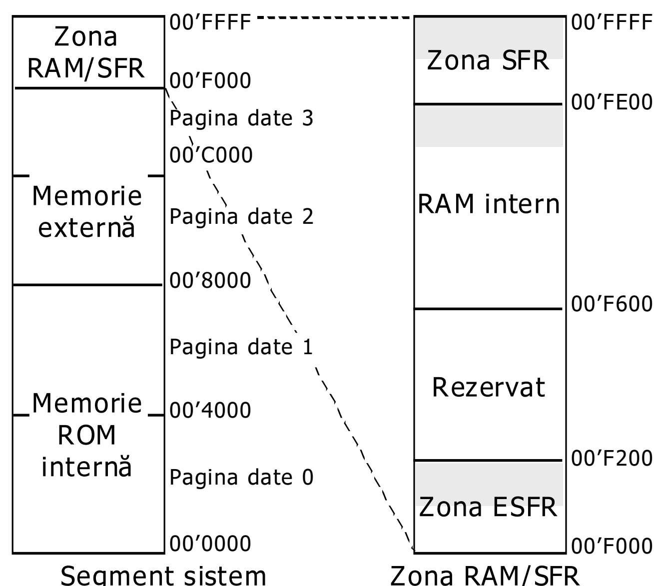


Figura 1.18. Harta memoriei circuitului 80C167

a) Stiva sistem

Stiva poate fi definită în memoria RAM la o adresă definită de registrul special SP. Mărimea stivei este controlată de biţii STKSZ din registrul special SYSCON, conform cu tabelul 2.1.

Tabelul 1.14

STKSZ	Mărime stivă (cuvinte)	Adresă stivă
000	256	00’FBFE-00’FA00 (implicit)
001	128	00’FBFE-00’FB00
010	64	00’FBFE-00’FB80
011	32	00’FBFE-00’FBC0
100	512	00’FBFE-00’F800
101, 110	Rezervat	
111	1024	00’FDFE-00’F600

Stiva sistem mai are două registre speciale STKUN şi STKOV utilizate pentru controlul limitei inferioare, respectiv superioare. Aceste două registre pot fi folosite nu numai pentru evitarea distrugerii datelor din stivă, dar permit şi implementarea unei stive circulare.

b) Registrele de uz general (GPR)

Registrele de uz general constau într-un bloc consecutiv de 16 locaţii de memorie (fiecare a câte 16 biţi) oriunde în interiorul memoriei RAM interne.

Registrul special CP (*context pointer* – indicator context) defineşte adresa de bază pentru bancul curent de registre, care constă în 16 registre de 16 biţi (de la R0 la R15) şi 16 registre de 8 biţi (de la RL0, RH0 la RL7, RH7). Registrele GPR de 8 biţi se suprapun cu primele opt registre de 16 biţi (R0…R7).

Registrele GPR sunt adresate pe 2, 4 sau 8 biţi, folosind ca registru de bază CP (independent de valoarea registrelor DPPx). Cele 16 registre sunt adresabile şi la nivel de bit.

Familia 80C16x permite comutarea rapidă a contextului, în fapt a registrului CP, permiţând existenţa simultană în memorie a mai multe bancuri de registre, chiar suprapuse parţial, dar numai cel adresat de registrul CP este activ.

c) Indicatorii pentru Interfaţa pentru evenimente de la periferice

Aceşti indicatori constau în 16 locaţii (fiecare a câte 16 biţi) poziţionate în memoria RAM internă la adrese de la 00'FCE0h la 00'FCEEh. Indicatorii sunt folosiţi pentru transferul datelor sub controlul interfeţei PEC. Fiecare din cele 8 canale ale PEC utilizează o pereche de indicatori memoraţi în două locaţii consecutive, indicatorul sursă (SRCP – *source PEC*), respectiv indicatorul destinaţie (DSTP – *destination PEC*).

În momentul în care este stabilit un transfer de date sub controlul PEC, indicatorii din perechea de registre SRCP şi DSTP permit transferul unor blocuri de date, de la adresa memorată în SRCP la adresa din DSTP.

d) Registrele speciale (SFR)

Unitatea centrală, interfeţele cu magistralele, porturile de intrare-ieşire şi a modulele interne sunt controlate prin intermediul unor registre speciale SFR. Aceste registre sunt dispuse în două blocuri, fiecare a câte 512 octeţi. Primul bloc, zona SFR, este dispus în memoria RAM la adresele 00'FFFFh…00'FE00h; al doilea bloc, zona ESFR (*extended SFR*) se găseşte la adresele 00'F1FFh…00'F000h.

Registrele speciale pot fi adresate indirect sau pe 16 biţi. Folosind un deplasament de 8 biţi împreună cu o adresă de bază implicită, se pot adresa cuvintele SFR sau octeţii inferiori ai acestora.

Atenţie! Modificarea oricărui octet din SFR are ca efect ştergerea octetului neadresat.

Jumătatea superioară a fiecărui bloc de registre speciale este adresabilă la nivel de bit, permiţând modificarea sau verificarea facilă a indicatorilor de stare ori control pentru modulele interne.

Pentru adresarea registrelor din zona ESFR folosind o adresă pe 8 biţi sau adresarea directă la nivel de bit, este obligatorie utilizarea unei instrucţiuni speciale, EXTR, instrucţiune care permite comutarea mecanismului de adresare din zona SFR în zona ESFR. Totuşi, pentru

adresarea pe 16 biţi sau indirectă, această instrucţiune nu este necesară. De asemenea, registrele GPR R0…R15 sunt duplicate, astfel încât adresarea lor este posibilă în orice mod fără a necesita comutarea respectivă.

De exemplu:

```
EXTR     #4                    ;comută ESFR următoarele 4 instrucţiuni
MOV      ODP2,#data16          ;ODP2 foloseşte adresare pe 8 biţi
BFLDL    DP6,#mask,#data8      ;Adresare pe bit pentru câmpuri de biţi
BSET     DP1H.7                ;Adresare pe bit
MOV      T8REL,R2              ;T8REL foloseşte adresare pe 16 biţi,
                               ;R2 este duplicat
…        …                     ;EXTR #4 îşi termină efectul
```

Pentru a minimiza folosirea instrucţiunii EXTR, zona ESFR conţine, de regulă, registre utilizate în principal la iniţializarea sistemului.

Registrele speciale utilizate de circuitul 80C167 sunt prezentate în tabelul 2.2. (unde *b* reprezintă registru adresabil la nivel de bit iar *E* semnifică zona ESFR), o descriere detaliată a fiecăruia găsindu-se la prezentarea fiecărui modul intern. Este recomandabilă utilizarea exactă a numelor prezentate în tabel întrucât programele de dezvoltare (compilatoare, assembler etc.) folosesc aceste mnemonice.

Tabelul 1.15

Nume	Adresă	Descriere
ADCIC b	FF98h	Registru control sfârşit conversie ADC
ADCON b	FFA0h	Registru control convertor ADC
ADDAT	FEA0h	Registru rezultat convertor ADC
ADDAT2	F0A0h **E**	Registru 2 rezultat convertor ADC
ADDRSEL1	FE18h	Registru 1 selectare adresă
ADDRSEL2	FE1Ah	Registru 2 selectare adresă
ADDRSEL3	FE1Ch	Registru 3 selectare adresă
ADDRSEL4	FE1Eh	Registru 4 selectare adresă
ADEIC b	FF9Ah	Registru control întrerupere convertor ADC
BUSCON0 b	FF0Ch	Registru 0 configurare bus
BUSCON1 b	FF14h	Registru 1 configurare bus
BUSCON2 b	FF16h	Registru 2 configurare bus
BUSCON3 b	FF18h	Registru 3 configurare bus
BUSCON4 b	FF1Ah	Registru 4 configurare bus
CAPREL	FE4Ah	Registru timer 2 captură/reîncărcare
CC0	FE80h	Registru CAPCOM 0
CC0IC b	FF78h	Registru control întrerupere CAPCOM 0
CC1	FE82h	Registru CAPCOM 1
CC1IC b	FF7Ah	Registru control întrerupere CAPCOM 1
CC2	FE84h	Registru CAPCOM 2
CC2IC b	FF7Ch	Registru control întrerupere CAPCOM 2
CC3	FE86h	Registru CAPCOM 3
CC3IC b	FF7Eh	Registru control întrerupere CAPCOM 3
CC4	FE88h	Registru CAPCOM 4
CC4IC b	FF80h	Registru control întrerupere CAPCOM 4
CC5	FE8Ah	Registru CAPCOM 5
CC5IC b	FF82h	Registru control întrerupere CAPCOM 5
CC6	FE8Ch	Registru CAPCOM 6
CC6IC b	FF84h	Registru control întrerupere CAPCOM 6
CC7	FE8Eh	Registru CAPCOM 7
CC7IC b	FF86h	Registru control întrerupere CAPCOM 7
CC8	FE90h	Registru CAPCOM 8
CC8IC b	FF88h	Registru control întrerupere CAPCOM 8
CC9	FE92h	Registru CAPCOM 9

CC9IC b	FF8Ah	Registru control întrerupere CAPCOM 9
CC10	FE94h	Registru CAPCOM 10
CC10IC b	FF8Ch	Registru control întrerupere CAPCOM 10
CC11	FE96h	Registru CAPCOM 11
CC11IC b	FF8Eh	Registru control întrerupere CAPCOM 11
CC12	FE98h	Registru CAPCOM 12
CC12IC b	FF90h	Registru control întrerupere CAPCOM 12
CC13	FE9Ah	Registru CAPCOM 13
CC13IC b	FF92h	Registru control întrerupere CAPCOM 13
CC14	FE9Ch	Registru CAPCOM 14
CC14IC b	FF94h	Registru control întrerupere CAPCOM 14
CC15	FE9Eh	Registru CAPCOM 15
CC15IC b	FF96h	Registru control întrerupere CAPCOM 15
CC16	FE60h	Registru CAPCOM 16
CC16IC b	F160h E	Registru control întrerupere CAPCOM 16
CC17	FE62h	Registru CAPCOM 17
CC17IC b	F162h E	Registru control întrerupere CAPCOM 17
CC18	FE64h	Registru CAPCOM 18
CC18IC b	F164h E	Registru control întrerupere CAPCOM 18
CC19	FE66h	Registru CAPCOM 19
CC19IC b	F166h E	Registru control întrerupere CAPCOM 19
CC20	FE68h	Registru CAPCOM 20
CC20IC b	F168h E	Registru control întrerupere CAPCOM 20
CC21	FE6Ah	Registru CAPCOM 21
CC21IC b	F16Ah E	Registru control întrerupere CAPCOM 21
CC22	FE6Ch	Registru CAPCOM 22
CC22IC b	F16Ch E	Registru control întrerupere CAPCOM 22
CC23	FE6Eh	Registru CAPCOM 23
CC23IC b	F16Eh E	Registru control întrerupere CAPCOM 23
CC24	FE70h	Registru CAPCOM 24
CC24IC b	F170h E	Registru control întrerupere CAPCOM 24
CC25	FE72h	Registru CAPCOM 25
CC25IC b	F172h E	Registru control întrerupere CAPCOM 25
CC26	FE74h	Registru CAPCOM 26
CC26IC b	F174h E	Registru control întrerupere CAPCOM 26
CC27	FE76h	Registru CAPCOM 27
CC27IC b	F176h E	Registru control întrerupere CAPCOM 27
CC28	FE78h	Registru CAPCOM 28
CC28IC b	F178h E	Registru control întrerupere CAPCOM 28
CC29	FE7Ah	Registru CAPCOM 29
CC29IC b	F184h E	Registru control întrerupere CAPCOM 29
CC30	FE7Ch	Registru CAPCOM 30
CC30IC b	F18Ch E	Registru control întrerupere CAPCOM 30
CC31	FE7Eh	Registru CAPCOM 31
CC31IC b	F194h E	Registru control întrerupere CAPCOM 31
CCM0 b	FF52h	Registru control mod CAPCOM 0
CCM1 b	FF54h	Registru control mod CAPCOM 1
CCM2 b	FF56h	Registru control mod CAPCOM 2
CCM3 b	FF58h	Registru control mod CAPCOM 3
CCM4 b	FF22h	Registru control mod CAPCOM 4
CCM5 b	FF24h	Registru control mod CAPCOM 5
CCM6 b	FF26h	Registru control mod CAPCOM 6
CCM7 b	FF28h	Registru control mod CAPCOM 7
CP	FE10h	Registru indicator context CPU
CRIC b	FF6Ah	Registru control întrerupere T2 CAPREL
CSP	FE08h	Registru indicator segment cod
DP0L b	F100h E	Registru control direcţie P0L

DP0H b	F102h E	Registru control direcţie P0H
DP1L b	F104h E	Registru control direcţie P1L
DP1H b	F106h E	Registru control direcţie P1H
DP2 b	FFC2h	Registru control direcţie P2
DP3 b	FFC6h	Registru control direcţie P3
DP4 b	FFCAh	Registru control direcţie P4
DP6 b	FFCEh	Registru control direcţie P6
DP7 b	FFD2h	Registru control direcţie P7
DP8 b	FFD6h	Registru control direcţie P8
DPP0	FE00h	Registru indicator pagină date 0 (10 biţi)
DPP1	FE02h	Registru indicator pagină date 1 (10 biţi)
DPP2	FE04h	Registru indicator pagină date 2 (10 biţi)
DPP3	FE06h	Registru indicator pagină date 3 (10 biţi)
EXICON b	F1C0h E	Registru control întrerupere externă
MDC b	FF0Eh	Registru control înmulţire-împărţire
MDH	FE0Ch	Registru control înmulţire-împărţire (cuvânt superior)
MDL	FE0Eh	Registru control înmulţire-împărţire (cuvânt inferior)
ODP2 b	F1C2h E	Registru control drenă în gol P2
ODP3 b	F1C6h E	Registru control drenă în gol P3
ODP6 b	F1CEh E	Registru control drenă în gol P6
ODP7 b	F1D2h E	Registru control drenă în gol P7
ODP8 b	F1D6h E	Registru control drenă în gol P8
ONES	FF1Eh	Registru valoare constantă 1
P0L b	FF00h	Registru inferior P0
P0H b	FF02h	Registru superior P0
P1L b	FF04h	Registru inferior P1
P1H b	FF06h	Registru superior P1
P2 b	FFC0h	Registru Port2
P3 b	FFC4h	Registru Port3
P4 b	FFC8h	Registru Port4 (8 biţi)
P5 b	FFA2h	Registru Port5
P6 b	FFCCh	Registru Port6 (8 biţi)
P7 b	FFD0h	Registru Port7 (8 biţi)
P8 b	FFD4h	Registru Port8 (8 biţi)
PECC0	FEC0h	Registru control canal 0 PEC
PECC1	FEC2h	Registru control canal 1 PEC
PECC2	FEC4h	Registru control canal 2 PEC
PECC3	FEC6h	Registru control canal 3 PEC
PECC4	FEC8h	Registru control canal 4 PEC
PECC5	FECAh	Registru control canal 5 PEC
PECC6	FECCh	Registru control canal 6 PEC
PECC7	FECEh	Registru control canal 7 PEC
PP0	F038h E	Registru perioadă PWM0
PP1	F03Ah E	Registru perioadă PWM1
PP2	F03Ch E	Registru perioadă PWM2
PP3	F03Eh E	Registru perioadă PWM3
PSW b	FF10h	Registru stare program
PT0	F030h E	Numărător sus/jos PWM0
PT1	F032h E	Numărător sus/jos PWM1
PT2	F034h E	Numărător sus/jos PWM2
PT3	F036h E	Numărător sus/jos PWM3
PW0	FE30h	Registru durată impuls PWM0
PW1	FE32h	Registru durată impuls PWM1
PW2	FE34h	Registru durată impuls PWM2
PW3	FE36h	Registru durată impuls PWM3
PWMCON0 b	FF30h	Registru 0 control modul PWM
PWMCON1 b	FF32h	Registru 1 control modul PWM

PWMIC b	F17Eh E	Registru control întreruperi modul PWM
RP0H b	F108h E	Registru configurare sistem la iniţializare
S0BG	FEB4h	Registru reîncărcare generator rată transmisie ASC0
S0CON b	FFB0h	Registru control ASC0
S0EIC b	FF70h	Registru control întrerupere eroare ASC0
S0RBUF	FEB2h	Registru bufer recepţie ASC0
S0RIC b	FF6Eh	Registru control întrerupere recepţie ASC0
S0TBIC b	F19Ch E	Registru control întrerupere bufer emisie ASC0
S0TBUF	FEB0h	Registru bufer emisie ASC0
S0TIC b	FF6Ch	Registru control întrerupere emisie ASC0
SP	FE12h	Registru indicator stivă
SSCBR	F0B4h E	Registru rată transmisie SSC
SSCCON b	FFB2h	Registru control SSC
SSCEIC b	FF76h	Registru control întrerupere eroare SSC
SSCRB	F0B2h E	Bufer SSC recepţie
SSCRIC b	FF74h	Registru control întrerupere recepţie SSC
SSCTB	F0B0h E	Bufer emisie SSC
SSCTIC b	FF72h	Registru control întrerupere emisie SSC
STKOV	FE14h	Registru depăşire superioară stivă
STKUN	FE16h	Registru depăşire inferioară stivă
SYSCON b	FF12h	Registru configurare sistem
T0	FE50h	Registru CAPCOM T0
T01CON b	FF50h	Registru control CAPCOM T0 şi T1
T0IC b	FF9Ch	Registru control întrerupere CAPCOM T0
T0REL	FE54h	Registru reîncărcare CAPCOM T0
T1	FE52h	Registru CAPCOM T1
T1IC b	FF9Eh	Registru control întrerupere CAPCOM T1
T1REL	FE56h	Registru reîncărcare CAPCOM T1
T2	FE40h	Registru T2
T2CON b	FF40h	Registru control T2
T2IC b	FF60h	Registru control întrerupere T2
T3	FE42h	Registru T3
T3CON b	FF42h	Registru control T3
T3IC b	FF62h	Registru control întrerupere T3
T4	FE44h	Registru T4
T4CON b	FF44h	Registru control T4
T4IC b	FF64h	Registru control întrerupere T4
T5	FE46h	Registru T5
T5CON b	FF46h	Registru control T5
T5IC b	FF66h	Registru control întrerupere T5
T6	FE48h	Registru T6
T6CON b	FF48h	Registru control T6
T6IC b	FF68h	Registru control întrerupere T6
T7	F050h E	Registru CAPCOM T7
T78CON b	FF20h	Registru control CAPCOM T7 şi T8
T7IC b	F17Ah E	Registru control întrerupere CAPCOM T7
T7REL	F054h E	Registru reîncărcare CAPCOM T7
T8	F052h E	Registru CAPCOM T8
T8IC b	F17Ch E	Registru control întrerupere CAPCOM T8
T8REL	F056h E	Registru reîncărcare CAPCOM T8
TFR b	FFACh	Registru indicator Trap
WDT	FEAEh	Registru timer watchdog
WDTCON	FFAEh	Registru control timer watchdog
XP0IC b	F186h E	Registru control întrerupere periferic X-BUS 0
XP1IC b	F18Eh E	Registru control întrerupere periferic X-BUS 1
XP2IC b	F196h E	Registru control întrerupere periferic X-BUS 2

XP3IC b	F19Eh E	Registru control întrerupere periferic X-BUS 3
ZEROS b	FF1Ch	Registru valoare constantă 0

e) Memoria externă

Circuitul 80C167 este capabil să folosească un spaţiu de memorie de până la 16 MB. Memoria externă este adresată numai prin intermediul interfeţei cu magistrala externă.

Funcţie de registrele de control ale unităţii centrale, sunt suportate patru mărimi ale bancurilor de memorie:

- mod nesegmentat: 64 kB cu A15…A0 pe porturile P1 sau P0;
- mod segmentat pe 2 biţi: 256 kB cu A17…A16 pe P4 şi A15…A0 pe P1 sau P0;
- mod segmentat pe 4 biţi: 1 MB cu A19…A16 pe P4 şi A15…A0 pe P1 sau P0;
- mod segmentat pe 8 biţi: 16 MB cu A23…A16 pe P4 şi A15…A0 pe P1 sau P0.

Fiecare banc poate fi adresat direct prin intermediul magistralei de adrese, folosind eventual pentru selectare circuitelor semnalele produse de circuit.

De asemenea, circuitul 80C167 suportă patru tipuri de magistrală:

- magistrală multiplexată pe 16 biţi, cu adrese şi date pe portul P0;
- magistrală multiplexată pe 8 biţi, cu adrese şi date pe portul P0/P0L;
- magistrală demultiplexată pe 16 biţi, cu adrese pe P1 şi date pe P0;
- magistrală demultiplexată pe 8 biţi, cu adrese pe P1 şi date pe P0L.

Modelul de memorie şi modelul de magistrală pot fi setate la iniţializarea circuitului, funcţie de starea pinilor $\overline{EA}$ şi portului P0 sau, ulterior, pot fi modificaţi prin program.

Datele, atât octeţi cât şi cuvinte, pot fi adresate numai prin intermediul adresării indirecte sau pe 16 biţi folosind unul din cele patru registre DPP. Orice acces la un cuvânt este făcut numai la o adresă pară.

Pentru transferul de date sub controlul PEC, memoria externă din segmentul 0 poate fi adresată indiferent de conţinutul registrelor DPP, numai prin intermediul registrelor sursă şi destinaţie.

Memoria externă nu este adresabilă la nivel de bit.

1.14. Unitatea centrală

Sarcinile de bază ale unităţii centrale sunt de a-i furniza instrucţiuni, de a le decodifica, de a asigura operanzi pentru unitatea aritmetică şi logică (ALU), precum şi de a memora rezultatele operaţiilor.

Schema bloc a unităţii centrale este prezentată în figura 2.5.

Semnificaţia blocurilor din structura unităţii centrale vor fi prezentate în paragrafele care urmează.

În timp ce accesul la memoria internă este asigurat chiar de procesor însuşi, controlul perifericelor şi memoriei externe este făcut prin intermediul unui bloc separat, interfaţa cu magistrala externă. Această modalitate permite procesorului să lucreze în timp ce, un acces la memoria externă, de regulă mai lentă, este în curs. Detalii suplimentare sunt prezentate în paragraful 1.15, Interfaţa cu magistrala externă (EBC).

Modulele interne ale circuitului lucrează aproape independent faţă de unitatea centrală, având disponibile un generator de ceas separat. Controlul şi schimbul de date între module şi unitatea centrală se realizează prin intermediul unor registre speciale (paragraful d).

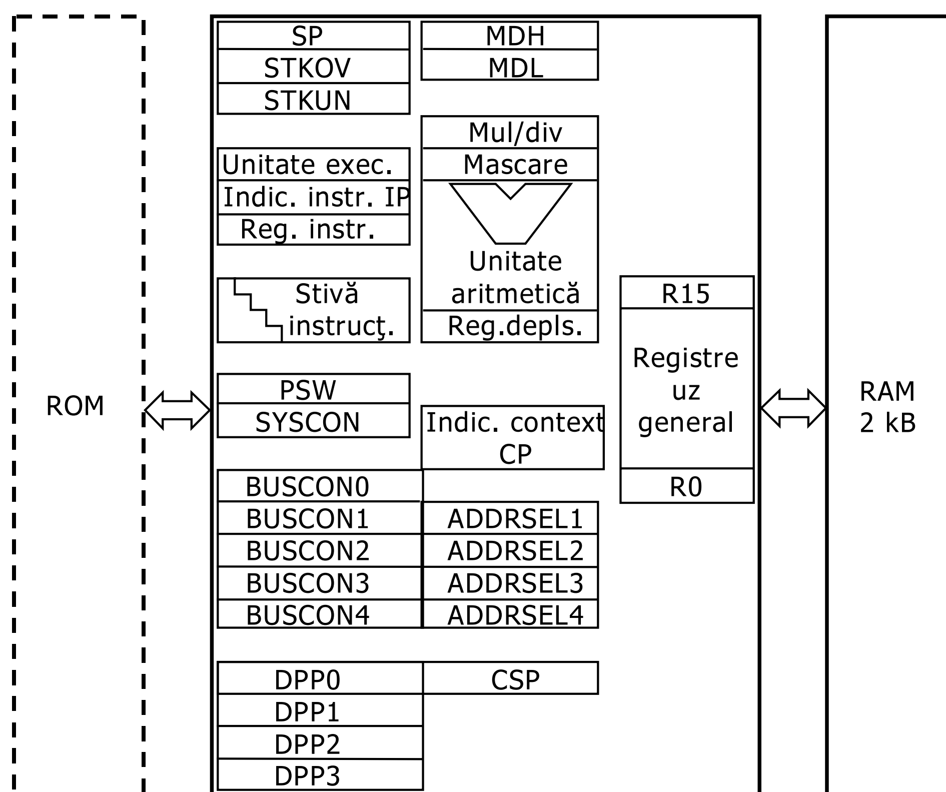


Figura 1.19. Unitatea centrală a microcontrolerului 80C167

În momentul în care anumite periferice necesită o acţiune specifică a unităţii centrale, un controler de întreruperi verifică toate cererile existente de la modulele interne sau externe şi, funcţie de priorităţile acestora, dacă sunt mai ridicate decât a instrucţiunii curente a unităţii centrale, este iniţiată o procedură de întrerupere care tratează evenimentul cu prioritatea cea mai ridicată (paragraful 1.16, Sistemul de întreruperi şi registrele PEC).

Un set de registre speciale controlează funcţionarea nucleului unităţii centrale:

- SYSCON, RP0H – configurare generală;
- PSW – stare şi control unitate centrală;
- IP,CSP – acces coduri;
- DPP0…DPP3 – control paginare date;
- CP – control acces registre uz general;

- SP,STKUN,STKOV – control stivă sistem;
- MDL,MDH,MDC – registre pentru înmulţire şi împărţire;
- ZEROS,ONES – registre cu valori constante.

1.14.1 Stiva de instrucţiuni

Deoarece circuitul 80C167 are o stivă cu patru nivele, până la patru instrucţiuni pot fi executate simultan. Această facilitate a unităţii centrale asigură un timp de execuţie a majorităţii instrucţiunilor într-un singur ciclu maşină (50 ns pentru frecvenţa de ceas de 40 MHz).

Cele patru nivele ale stivei de instrucţiuni au următoarele funcţiuni:

1. FETCH (aducere) – în acest nivel este asigurată aducerea instrucţiunilor, adresate prin registrele IP şi CSP, din memoria internă sau externă, RAM ori ROM, în unitatea centrală;
2. DECODE (decodificare) – acest nivel asigură decodificarea instrucţiunilor şi, dacă este necesar, este calculată adresa operandului şi acesta este încărcat. Pentru instrucţiunile de salt, registrele IP şi CSP sunt actualizate cu valoarea destinaţiei saltului. În operaţiunile care implică utilizarea stivei sistem, registrul SP este incrementat sau decrementat.
3. EXECUTE (execuţie) – în acest nivel, operaţia încărcată anterior în ALU, este executată. Suplimentar, sunt actualizaţi indicatorii din registrul de stare PSW funcţie de rezultat. De asemenea, modificarea registrelor SFR şi incrementarea sau decrementarea registrelor de uz general folosite pentru adresare indirectă sunt efectuate în acest nivel.
4. WRITE BACK (rescriere) – în acest nivel, toţi operanzii externi şi cei rămaşi în memoria RAM în urma instrucţiunii curente sunt reactualizaţi.

O particularitate a circuitului 80C167 sunt aşa numitele *instrucţiuni inserate*. Aceste instrucţiuni sunt introduse automat de unitatea centrală în situaţia în care survine o instrucţiune care nu poate fi executată într-un singur ciclu maşină. Această inserare este efectuată în nivelul 2 (decodificare), ulterior, procedura fiind asemănătoare cu instrucţiunile normale. De asemenea, întreruperile sunt prelucrate prin acelaşi tip de inserare de ciclu maşină.

Fiecare instrucţiune singulară trebuie să treacă prin cele patru nivele ale stivei de instrucţiuni, indiferent dacă fiecare nivel al stivei efectuează ceva sau nu. Deoarece parcurgerea unui nivel din stiva de instrucţiuni necesită un ciclu maşină, orice instrucţiune de sine stătătoare are obligatoriu patru cicluri maşină. Implementarea stivei de instrucţiuni permite astfel execuţia simultană a patru instrucţiuni, fiecare instrucţiune fiind executată de patru ori mai repede, într-un singur ciclu maşină.

Bineînţeles, în situaţia existenţei unui salt, stiva de instrucţiuni trebuie reîncărcată. Totuşi, mecanismul *inserare instrucţiune* este optimizat, astfel că nu întârzie fluxul execuţiei decât cu un singur ciclu maşină. Mai mult decât atât, dacă saltul face parte dintr-o buclă, unitatea centrală are un mecanism

suplimentar, saltul *cache*. Procedura constă în memorarea locaţiei destinaţiei de salt într-o locaţie specială (*cache*) într-un ciclu inserat, la reluarea buclei eliminându-se ciclurile suplimentare de inserare.

Deoarece patru instrucţiuni diferite sunt prelucrate simultan apar, totuşi unele probleme care trebuiesc avute în vedere la realizarea aplicaţiilor:

a) Actualizarea indicatorului context

O instrucţiune care calculează adresa unui operand din registrele generale de lucru prin intermediul registrului CP, de regulă nu este capabilă să folosească noua valoare a registrului CP imediat după încheierea instrucţiunii. Pentru a nu fi produse rezultate neaşteptate, este indicat ca după actualizarea registrului CP să urmeze măcar o instrucţiune care să nu facă apel la registrele de uz general, ca în exemplul următor în scris în assembler:

```
In        SCXT CP,#0FC00h      ;selectarea unui nou context
In+1      …                    ;instrucţiuni de protecţie
In+2      MOV R0,#data         ;instrucţiuni care folosesc noul context.
```

b) Actualizarea indicatorului paginii de date

O instrucţiune care calculează adresa unui operand prin intermediul registrelor DPP, de regulă nu este capabil să folosească noua valoare a registrelor DPP imediat după încheierea instrucţiunii. Pentru a nu fi produse rezultate neaşteptate, este indicat ca după actualizarea registrelor DPP să urmeze măcar o instrucţiune care să nu folosească pentru adresare indirectă sau lungă registrele DPP, ca în exemplul următor în scris în assembler:

```
In        MOV DPP0,#4          ;selectează pagina 4 date
In+1      …                    ;instrucţiuni de protecţie
In+2      MOV DPP0:0000h,R0    ;mută conţinutul registrului general R0 la adresa
                               ;01’0000h (în pagina de date 4).
```

c) Actualizarea explicită a indicatorului stivei

Nici una din instrucţiunile RET, RETI, RETS, RETP sau POP nu este capabil să folosească noua valoare a registrului SP imediat după actualizarea acestuia. Pentru a nu fi produse rezultate neaşteptate, este indicat ca după actualizarea explicită a registrului SP să urmeze măcar o instrucţiune care să nu folosească stiva, ca în exemplul următor în scris în assembler:

```
In        MOV SP,#0FA40h      ;selectează vârful stivei
In+1      …                   ;instrucţiuni de protecţie
In+2      POP,R0              ;încarcă registrul general R0 cu valoarea aflată
                              ;în vârful stivei.
```

d) Controlul întreruperilor

Modificările prin program (implicite sau explicite ale registrului PSW) sunt realizate în ciclul maşină 3 (execuţie). Păstrarea unei viteze ridicate de tratare a întreruperilor a condus la necesitatea adoptării unei timp scurt de reacţie a acestui sistem, astfel încât o cerere de întrerupere poate fi acceptată în timpul sau după instrucţiunea imediat următoare blocării sistemului de întreruperi, ca în exemplul următor scris în assembler:

```
In        BCLR IEN   ;inactivare generală întreruperi
In+1      …          ;instrucţiune intreruptibilă
In+2                 ;instrucţiune neinteruptibilă
```

```
                …
In+k      BSET IEN    ;revalidare întreruperi
In+k+1    …           ;instrucţiune neinteruptibilă
In+k+2    …           ;instrucţiune intreruptibilă
```

e) Iniţializarea porturilor de intrare-ieşire

Modificarea direcţiei pinului unui port (intrare sau ieşire) devine efectivă abia după următoarea instrucţiune. Este obligatoriu ca după iniţializarea unui port, următoarea comandă să nu se refere la acelaşi port, ca în exemplul următor scris în assembler:

Eronat:

```
In        BSET DP7.5          ;setează pinul P7.5 ca ieşire;
In+1      BSET P7.0           ;mod eronat de setare a portului P3
```

Corect:

```
In        BSET DP7.5          ;setează pinul P7.5 ca ieşire;
In+1      NOP                 ;sau altă instrucţiune care nu afectează P3
In+2      BSET P7.0           ;mod corect de setare a portului P3
```

f) Schimbarea configuraţiei sistemului

Instrucţiunile care urmează unei comenzi de schimbare a configuraţiei sistemului prin intermediul registrelor SYSCON, BUSCON şi ADDRSEL (de exemplu segmentări, mărime stivă, alocare memorie internă, caracteristici memorie adresată etc.) nu pot utiliza imediat resursele definite. Ca regulă obligatorie, accesul codului de la noua zonă ROM este posibilă numai după executarea unui salt absolut în zona respectivă.

1.14.2 Timpul de execuţie al instrucţiunilor

De regulă, timpul de execuţie al unei instrucţiuni depinde de unde este adusă instrucţiunea şi, eventual, de unde sunt citiţi sau scrişi operanzii. Modul cel mai rapid de lucru permis de circuitul 80C167 este atins dacă programul este înregistrat în memoria ROM internă. În acest caz, majoritatea instrucţiunilor sunt executate într-un ciclu maşină care, oricum, este timpul minim de execuţie al unei instrucţiuni.

Accesul la memoria externă nu încetineşte excesiv lucrul circuitului, întrucât interfaţa cu magistrala externă care controlează dispozitivele externe lucrează în paralel cu unitatea centrală.

De asemenea, durata de execuţie a programelor din memoria externă mai este influenţată de modelul magistralei externe, timpii de aşteptare programaţi etc.

Câteva instrucţiuni ale circuitului 80C167 au, prin definiţie, o viteză mai mică de execuţie; acestea sunt: unele tipuri de salturi, înmulţirea, împărţirea şi o instrucţiune specială de mutare.

Orientativ, în tabelul 2.3. sunt prezentaţi timpii de execuţie (în ns) pentru un microcontroler cu ceas de 20 MHz.

Tabelul 1.16

Zona memorie	Încărcare instrucţiune		Operand 16 biţi	
	Instrucţiune 16 biţi	Instrucţiune 32 biţi	Citire	Scriere
ROM intern	100	100	100	–
RAM intern	300	400	0/50	0
Bus demultiplexat 16 biţi	100	200	100	100
Bus multiplexat 16 biţi	150	300	150	150

Bus demux. 8 biţi	200	400	200	200
Bus multiplexat 8 biţi	300	600	300	300

Funcţie de modul de dezvoltare al unei aplicaţii, încărcarea programului din memoria RAM internă oferă flexibilitate în sensul modificărilor necesare pentru punerea la punct a aplicaţiei, în timp ce, încărcarea codului din memoria ROM internă este recomandată în final, după ce aplicaţia a fost pusă la punct.

Pentru a obţine o durată minimă, trebuie evitată folosirea următoarelor instrucţiuni:

- citirea operanzilor din memoria ROM internă;
- citirea operanzilor din memoria RAM internă prin intermediul unor instrucţiuni de adresare indirectă;
- citirea operanzilor din SFR imediat după scrierea lor;
- utilizarea operanzilor din memoria externă;
- salturi condiţionale imediat după scrierea registrului de stare PSW;
- salturi la adrese nealiniate din memoria ROM internă.

1.14.3 Registrele speciale ale unităţii centrale

Unitatea centrală necesită un set de registre speciale utilizate pentru păstrarea informaţiilor referitoare la starea sistemului, controlul sistemului şi configurarea magistralelor, segmentarea memoriei program, utilizarea paginilor de memorie de date, asigurarea unităţii aritmetice (ALU) cu constante de uz general sau operanzi pentru înmulţire ori împărţire, precum şi accesul la registrele de uz general şi la stiva sistem.

Accesul la registrele SFR ale unităţii centrale se face simplu, prin orice instrucţiune capabilă să adreseze spaţiul de memorie al SFR. Totuşi, pentru a asigura corecta funcţionare a circuitului, există unele restricţii, cum ar fi:

- indicatorul de instrucţiuni IP şi indicatorul segmentului de program CSP nu pot fi adresate direct dar pot fi modificate prin intermediul instrucţiunilor de salt;
- registrele PSW, SP şi MDH nu pot fi modificate explicit prin program dar, implicit, sunt actualizate de instrucţiunile executate.

Atenţie! Orice scriere a unui octet dintr-un registru SFR şterge octetul complementar neadresat.

Biţii SFR rezervaţi nu pot fi citiţi. La citire vor avea întotdeauna valoarea 0h.

Modificarea prin program a unui registru SFR este prioritară faţă de o actualizare simultană datorată modulelor interne.

a) Registrul de configurare a sistemului (SYSCON)

Acest registru, adresabil la nivel de bit, asigură configurarea generală şi a funcţiilor de control ale sistemului. Valoarea la iniţializare a registrului depinde de starea pinilor portului P0 şi a semnalului $\overline{EA}$ la momentul respectiv. Structura registrului SYSCON este prezentată în tabelul 2.4.

Tabelul 1.17

SYSCON (FF12h)	STKSZ			ROMS1	SGTDIS	ROMEN	BYTDIS	CLKEN	WRCFG	–	–	–	–	–	VISIBLE	XPER-SHARE

STKSZ	Selectează mărimea stivei sistem între 32 şi 1024 cuvinte.
ROMS1	0h: memoria ROM internă este în segmentul 0 (00'0000h...00'7FFFh); 1h: memoria ROM internă este în segmentul 1 (01'0000h...01'7FFFh).
SGTDIS	0h: segmentarea este validată (CSP este salvat în stivă); 1h: segmentarea este invalidată (CSP nu este salvat în stivă).
ROMEN	0h: memoria ROM internă dezafectată; 1h: memoria ROM internă validată.
BYTDIS	0h: pinul $\overline{BHE}$ validat; 1h: pinul $\overline{BHE}$ (P3.12) poate fi folosit ca intrare-ieşire de uz general.
CLKEN	0h: pinul CLKOUT (P3.15) poate fi intrare-ieşire de uz general; 1h: pinul CLKOUT validat; pe pin se regăseşte semnalul ceas sistem.
WRCFG	0h: pinii $\overline{WR}$ şi $\overline{BHE}$ funcţionează normal; 1h: pinul $\overline{WR}$ se comportă ca $\overline{WRL}$ iar $\overline{BHE}$ ca $\overline{WRH}$.
VISIBLE	0h: accesul la perifericele X-BUS este făcut intern; 1h: accesul la perifericele X-BUS este *vizibil* pe pinii circuitului.
XPER—SHARE	0h: accesul extern la perifericele X-BUS este dezactivat; 1h: accesul extern la perifericele X-BUS este permis în modul HOLD.

b) Registrul de stare a procesorului (PSW)

Acest registru, adresabil la nivel de bit, indică starea curentă a circuitului. El conţine două grupuri de biţi, unul descriind starea unităţii aritmetice iar celălalt controlând sistemul de întreruperi. Suplimentar, este prezent şi un bit, USR0, cu destinaţie generală. Structura registrului PSW este prezentată în tabelul 2.5.

Tabelul 1.18

PSW (FF10h)	ILVL				IEN	HLDEN	–	–	–	USR0	MULIP	E	Z	V	C	N

ILVL	Câmpul de patru biţi ILVL (*interrupt level*) defineşte nivelul de prioritate alocat procesorului, între 15 (când procesorul poate fi întrerupt numai de o întrerupere nemascabilă sau întrerupere generată de o excepţie hardware) şi 0 (prioritatea cea mai mică – poate fi întrerupt de orice eveniment extern).
IEN	IEN=1h sau IEN=0h validează sau invalidează global sistemul de întreruperi.
HLDEN	HLDEN=1h sau HLDEN=0h validează sau invalidează semnalele de arbitrare ale magistralei ($\overline{BREQ}$, $\overline{HLDA}$ şi $\overline{HOLD}$). Dacă bitul este şters, pinii corespunzători (P6.7...P6.5) pot fi utilizaţi ca intrări/ieşiri de uz general.
USR0	Indicator de uz general la dispoziţia programatorului.
MULIP	0h: nu există înmulţire/împărţire în curs de execuţie; 1h: operaţia de înmulţire/împărţire este întreruptă.
E	Este setat dacă operandul sursă al unei instrucţiuni este egal cu numărul cel mai mic (-8000h pentru cuvinte şi –80h pentru octeţi).
Z	Este setat dacă rezultatul operaţiei ALU este zero. Pentru operaţii booleene cu un singur operand, Z are valoarea negată a bitului respectiv, în timp ce pentru operaţii booleene cu doi operanzi capătă valoarea SAU-NU-LOGIC a celor doi biţi.

<table>
<tr><td rowspan="6">V</td><td colspan="3">Pentru adunare, scădere şi complementare faţă de 2, V indică o depăşire a domeniului maxim a numerelor cu semn (-8000h…+7FFFh pentru cuvinte, respectiv –80h…7Fh pentru octeţi). Pentru înmulţiri şi împărţiri V este setat dacă rezultatul nu poate fi reprezentat ca un cuvânt. Pentru operaţii booleene cu doi operanzi când capătă valoarea SAU-LOGIC a celor doi biţi. De asemenea, V este utilizat pentru operaţii de rotunjire pentru rotiri sticky bit, după cum urmează:</td></tr>
<tr><td>C</td><td>V</td><td>Rotunjire</td></tr>
<tr><td>0</td><td>0</td><td>Nu există eroare de rotunjire.</td></tr>
<tr><td>0</td><td>1</td><td>Eroare de rotunjire < ½ LSB.</td></tr>
<tr><td>1</td><td>0</td><td>Eroare de rotunjire = ½ LSB.</td></tr>
<tr><td>1</td><td>1</td><td>Eroare de rotunjire > ½ LSB</td></tr>
<tr><td>C</td><td colspan="3">După o operaţie de adunare, C indică generarea unui transport al bitului cel mai semnificativ al operandului (octet sau cuvânt). După o scădere sau comparare, C indică un împrumut. Pentru deplasări sau rotiri, C are valoarea ultimului bit deplasat. C este întotdeauna şters după operaţii logice, înmulţiri sau împărţiri, cu excepţia unei operaţii booleene cu doi operanzi când capătă valoarea ŞI-LOGIC a celor doi biţi.</td></tr>
<tr><td>N</td><td colspan="3">Este setat dacă cel mai semnificativ bit al rezultatului este 1h şi şters în caz contrar. Pentru operaţii cu numere întregi N poate fi interpretat ca bit de semn. Pentru operaţii booleene cu un singur operand, N are valoarea anterioară a bitului respectiv, în timp ce pentru operaţii booleene cu doi operanzi capătă valoarea SAU-EXCLUSIV a celor doi biţi.</td></tr>
</table>

c) Indicatorul de instrucţiuni (IP) şi indicatorul segmentului de program (CSP)

Registrul IP determină adresa pe 16 biţi a instrucţiunii curente încărcate din segmentul adresat de CSP. IP nu se regăseşte în spaţiul de memorie adresabil şi, deci, nu poate fi modificat direct de programator. Totuşi, IP poate fi modificat, indirect, prin intermediul stivei sistem sau a unui salt.

Registrul CSP selectează care segment este utilizat împreună cu IP pentru încărcarea instrucţiunilor.

Structura celor două registre este prezentată în tabelul 2.6.

Tabelul 1.19

IP (––––)	IP															
IP	Specifică adresa curentă din interiorul segmentului SEGNR.															
CSP (FE08h)	–	–	–	–	–	–	–	–	SEGNR							
SEGNR	Specifică segmentul de cod de unde sunt încărcate instrucţiunile. Dacă segmentarea este dezactivată, SEGNR este ignorat.															

Adresa instrucţiunii este calculată prin extinderea celor 16 biţi ai IP cu 2/4/8 biţi ai CSP, funcţie de modul de adresare selectat (figura 2.6).

În cazul lucrului cu memorie segmentată, biţii pentru selectarea segmentului de lucru (7…0/3…0/1…0) sunt generaţi pe pinii de adresă A23/A19/A17…A16 ai portului P4 pentru toate apelurile la memoria externă.

Pentru lucrul cu memorie nesegmentată sau în modul *Single Chip*, conţinutul CSP este nesemnificativ, întrucât memoria externă este limitată numai la segmentul 0, accesat numai prin IP.

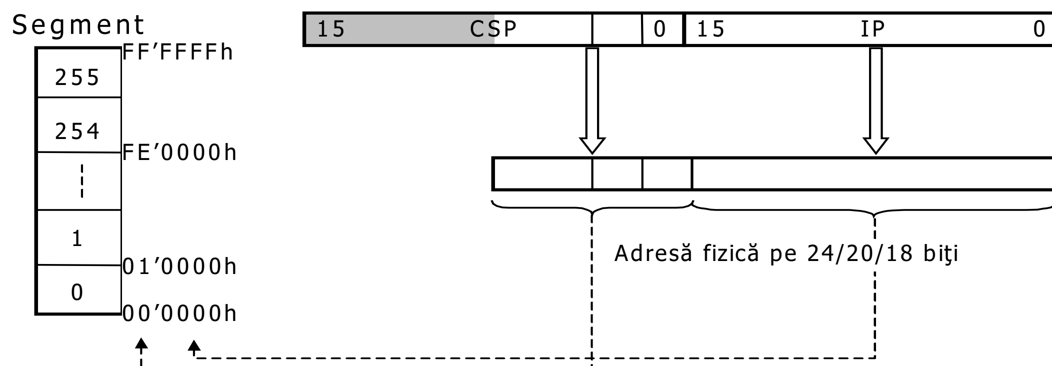


Figura 1.20. Generarea adresei cu IP şi CSP

d) Indicatorii paginilor de date (DPP0, DPP1, DPP2 şi DPP3)

Aceste patru registre selectează patru pagini de date diferite care sunt active simultan în timpul execuţiei programului. Primii 10 biţi mai puţin semnificativi selectează una din cele 1024 de pagini de 16 kB posibile. Astfel, DPP permit adresarea întregii memorii prin pagini de 16 kB fiecare.

Registrele DPP sunt folosite implicit în momentul oricărui acces la memorie prin intermediul instrucţiunilor indirecte sau pe 16 biţi (fac excepţie instrucţiunile EXTinse sau transferul de date prin intermediul PEC).

După iniţializare, cele patru registre sunt setate astfel încât permit adresarea directă a primelor pagini (3…0) din segmentul 0.

Structura registrelor DPPx este prezentată în tabelul 2.7.

Tabelul 1.20

DPP0 (FE00h)	–	–	–	–	–	DPP0PN
DPP1 (FE02h)	–	–	–	–	–	DPP1PN
DPP2 (FE04h)	–	–	–	–	–	DPP2PN
DPP3 (FE06h)	–	–	–	–	–	DPP3PN
DPPxPN	Specifică pagina de date selectată de registrul DPPxPN. Dacă segmentarea este dezactivată, doar primii doi biţi mai puţin semnificativi sunt reprezentativi.					

Principiul de lucru al paginării constă în concatenarea primilor 14 biţi ai unei adrese indirecte sau ai unei adrese pe 16 biţi cu conţinutul registrului DPPx selectat de primii 2 biţi ai adresei. Această modalitate este prezentată în figura 2.7.

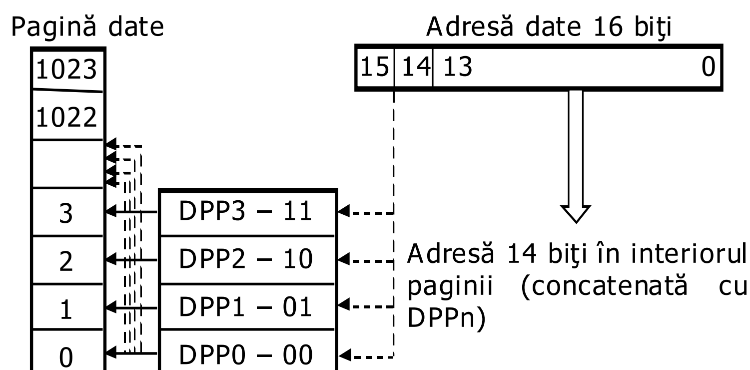


Figura 1.21. Generarea adresei cu DPP

Atenţie! În situaţia lucrului cu memorie nesegmentată, doar primii doi biţi ai DPP sunt folosiţi pentru generarea adresei. În această situaţie, valoarea registrelor DPP trebuie modificată cu grijă, pentru a nu se produce rezultate nedorite.

e) Indicatorul context (CP)

Acest registru este folosit pentru a selecta zona din memoria RAM internă alocată pentru cele 16 registre de uz general (GPR).

Structura registrului CP este prezentată în tabelul 2.8.

Tabelul 1.21

CP (FE10h)	1	1	1	1	CP											
CP	Specifică adresa de bază a bancului de registre GPR curent.															

Atenţie! Programatorul trebuie să fie circumspect cu declararea valorii CP astfel încât GPR să fie întotdeauna în memoria RAM internă. Acest fapt implică setarea CP în zona 00’F600h…00’FDFEh.

Câteva moduri de adresare folosesc implicit CP pentru calcularea adreselor. Acestea sunt:

- Adresare pe 4 biţi (mnemonic *Rw* sau *Rb*) – specifică o adresă relativă la locaţia de memorie indicată de CP. Funcţie de operand (cuvânt – *Rw*, respectiv octet – *Rb*), numărul registrului GPR este sau nu înmulţit cu 2 înainte de a fi adunat la conţinutul CP (figura 2.8.).

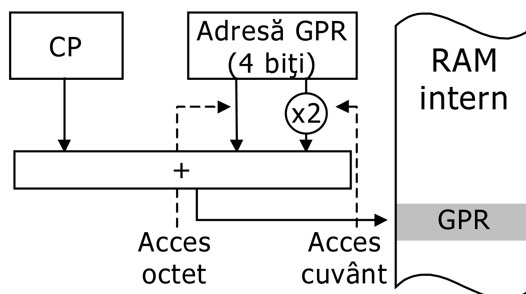


Figura 1.22. Calculul adresei GPR

- Adresare pe 2 biţi – pentru unele instrucţiuni numai primii doi biţi ai GPR sunt folosiţi pentru o adresare indirectă. Modul de calcul al adresei este identic cu cel pentru adresarea pe 4 biţi.
- Adresare pe 8 biţi (mnemonic *reg* sau *bitoff*) – dacă sunt în domeniul F0h-FFh, interpretează cei 4 biţi mai puţin semnificativi ca o adresă GPR pe 4 biţi, în timp ce ceilalţi biţi mai semnificativi sunt ignoraţi. Modul de calcul al adresei este identic cu cel pentru adresarea pe 4 biţi. Pentru acces la nivel de bit, adresa GPR este calculată ca mai sus iar poziţia bitului este specificată de ceilalţi 4 biţi mai semnificativi.

f) Indicatorii stivă sistem (SP), depăşire superioară stivă (STKOV) şi depăşire inferioară stivă (STKUN)

Indicatorul stivă sistem este utilizat pentru specificarea adresei curente a stivei sistem. Registrul SP este pre-decrementat când o dată este introdusă în stivă, respectiv post-incrementat când o dată este extrasă din stivă.

Deoarece bitul cel mai puţin semnificativ este şters şi biţii 15…12 sunt setaţi de către circuit, SP aparţine domeniului F000h…FFFEh, în memoria RAM internă. Prin program, cu ajutorul indicatorilor de depăşire STKOV şi STKUN, poate fi creată o stivă *virtuală*, de dimensiuni mai mari.

Indicatorul depăşire superioară STKOV este comparat cu valoarea SP la fiecare operaţiune care implică folosirea stivei şi, în cazul în care (SP)<(STKOV) este iniţiată o întrerupere.

Întreruperea poate fi tratată în două moduri:

- eroare fatală, situaţie în care este semnalat faptul că datele din partea inferioară a stivei sunt compromise;
- deplasarea automată a stivei de sistem pentru a permite folosirea stivei sistem ca *Stivă Cache*. În acest caz, registrul STKOV trebuie iniţializat cu o valoare care reprezintă noua adresă curentă a stivei sistem la care se adaugă 12 (cele şase cuvinte suplimentare adăugate sunt necesare pentru a preveni orice eroare care putea surveni datorită suprapunerii cu o întrerupere, pentru registrele salvate automat de întrerupere, PSW, IP şi CSP).

Indicatorul depăşire superioară STKUN este comparat cu valoarea SP la fiecare operaţiune care implică folosirea stivei şi, în cazul în care (SP)>(STKUN) este iniţiată o întrerupere.

Întreruperea poate fi tratată în două moduri, asemănător cu întreruperea generată de STKOV.

Atenţie! Dacă SP este modificat direct, prin intermediul unei instrucţiuni MOV şi noua valoare este în afara limitelor STKOV–STKUN, nu va fi generată o întrerupere.

Structura registrelor pentru controlul stivei sistem este descrisă în tabelul 2.9.

Tabelul 1.22

Registru						
SP (FE12h)	1	1	1	1	SP	1
STKOV (FE14h)	1	1	1	1	STKOV	1
STKUN (FE16h)	1	1	1	1	STKUN	1

SP	Specifică adresa curentă a stivei sistem.
STKOV	Specifică limita inferioară a stivei sistem.
STKUN	Specifică limita superioară a stivei sistem.

g) Registrele pentru înmulţire/împărţire (MDH, MDL şi MDC)

Registrele MDH (FE0Ch) şi MDL (FE0Eh) formează împreună un registru de 32 de biţi utilizat implicit de unitatea centrală pentru operaţiile de înmulţire şi împărţire.

După înmulţire, MDH conţine octetul superior al rezultatului iar MDL pe cel inferior.

Pentru împărţire, MDH este încărcat cu octetul superior iar MDL cu cel inferior al deîmpărţitului. După împărţire, MDH conţine restul iar MDL câtul.

Registrul de control MDC (FF0Eh) este modificat de unitatea centrală pe parcursul operaţiilor de înmulţire/împărţire. Singurul bit accesibil programatorului MDRIU (registru MD în uz – MDC.4) se setează după cum urmează:

- dacă registrele MDL şi MDC sunt modificate prin program sau o operaţie de înmulţire/împărţire este în curs, indicatorul este setat;
- dacă registrele MDL şi MDC sunt citite, indicatorul este şters.

h) Registrele constante (ZEROS şi ONES)

Aceste registre au valorile setate prin hardware. Registrul ZEROS (FF1Ch) are valoarea 00h în timp ce registrul ONES (FF1Eh) are valoarea FFh.

1.15. Interfaţa cu magistrala externă (EBC)

Cu toate că circuitul 80C167 este înzestrat cu un set puternic de periferice şi memorii interne, acestea acoperă numai o mică parte din spaţiul total de memorie de 16 MB. Rolul interfeţei cu magistrala externă este tocmai de a adresa periferice şi memorii externe.

Funcţiile interfeţei sunt asigurate de mai multe registre speciale şi, anume, SYSCON (prezentat la 1.14.3.a), BUSCONx (x=5…0), ADDRSELx (x=4…0), precum şi de o serie de pini dedicaţi sau cu funcţii alternative (porturile P0, P1, P3 şi P4 şi pinii ALE, $\overline{RD}$, $\overline{WR}$/$\overline{WRL}$, $\overline{BHE}$/$\overline{WRH}$, $\overline{EA}$, RSTIN şi $\overline{READY}$).

Registrele BUSCON descriu ciclurile magistralei externe funcţie de adrese (multiplexate sau demultiplexate), date (16 biţi sau 8 biţi), selecţie dispozitive, sincronizarea cu circuitele externe (stări de aşteptare, control $\overline{READY}$, întârzieri ALE şi $\overline{RD}$/$\overline{WR}$). Aceşti parametri sunt folosiţi pentru accesarea unei zone specifice de adrese definite prin registrul corespunzător ADDRSEL. Astfel, cele patru perechi BUSCON/ADDRSEL permit accesarea a

patru *ferestre de adrese* independente, în timp ce controlul accesului în afara acestor zone este realizat de registrul BUSCON0.

Dacă circuitul este singular, fără alte dispozitive externe, la iniţializare pinul $\overline{EA}$ trebuie să fie în starea 1 LOGIC. În această situaţie, orice acces la dispozitivele externe generează o întrerupere datorită excepţiei ILLBUS.

1.15.1 Modurile de lucru ale magistralei externe

a) Magistrală multiplexată

În modul multiplexat, adresa pe 16 biţi din segmentul curent şi datele folosesc portul P0. Adresele şi datele sunt multiplexate în timp, fiind separate în exterior. Registrul de separare are o lăţime dictată de mărimea magistralei de date (poate fi de 8 biţi sau 16 biţi). Adresele superioare, An…A16 sunt extrase în permanenţă pe portul P4; ele nu sunt multiplexate şi nu necesită registre de separare.

Interfaţa cu magistrala externă iniţiază un acces extern prin generarea semnalului ALE; frontul căzător al ALE comandă un registru extern de 16 biţi pentru capturarea adresei. După o perioadă de timp programabilă, adresa este înlăturată de pe magistrală. Acum, interfaţa cu magistrala externă activează un semnal de comandă ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$ sau $\overline{WRH}$). Datele sunt introduse pe magistrală fie de EBC (pentru cicluri de scriere), fie de dispozitivele externe (pentru cicluri de citire). După o perioadă de timp funcţie de viteza circuitelor externe, datele de pe magistrală sunt valide:

- ciclu de scriere – datele sunt încărcate şi semnalele de comandă sunt dezactivate; circuitul extern trece din nou în starea de înaltă impedanţă;
- ciclu de scriere – semnalele de comandă sunt dezactivate; datele rămân valide pe magistrală până la un nou ciclu extern.

Diagrama de timp este prezentată în figura 2.9.a.

b) Magistrală demultiplexată

În modul demultiplexat, adresa în interiorul segmentului este pe portul P1 iar datele folosesc portul P0 (pentru 16 biţi) sau P0L (pentru 8 biţi). Magistrala superioară de adrese sunt extrase în permanenţă pe portul P4.

EBC iniţiază un acces extern prin plasarea adresei pe magistrală. După o perioadă programabilă, EBC activează semnalele de comandă necesare ($\overline{RD}$, $\overline{WR}$, $\overline{WRL}$ sau $\overline{WRH}$). Datele sunt introduse pe magistrală fie de EBC (pentru cicluri de scriere), fie de dispozitivele externe (pentru cicluri de citire). După o perioadă de timp funcţie de viteza circuitelor externe, datele de pe magistrală sunt valide:

- ciclu de scriere – datele sunt încărcate şi semnalele de comandă sunt dezactivate; circuitul extern trece din nou în starea de înaltă impedanţă;
- ciclu de scriere – semnalele de comandă sunt dezactivate; datele rămân valide pe magistrală până la un nou ciclu extern.

Diagrama de timp este prezentată în figura 2.9.b.

c) Comutarea între tipurile de magistrală

EBC permite schimbarea dinamică a tipului magistralei, de exemplu în cazul a două cicluri externe consecutive. Unele zone de adrese pot folosi magistrale multiplexate sau demultiplexate, precum şi diferiţi timpi de întârziere, semnale de control $\overline{\text{READY}}$ etc. Schimbarea dinamică a magistralei externe poate fi făcută în două moduri:

- Reprogramarea registrelor BUSCON şi ADDRSEL permite fie schimbarea tipului de magistrală pentru o anumită zonă de memorie, fie schimbarea dimensiunii zonei de memorie unde este aplicabil tipul respectiv de magistrală. Reprogramarea permite folosirea unui număr de ferestre cu parametri diferiţi mai mare decât cele disponibile prin BUSCONx.
- Comutarea între ferestre diferite selectează automat tipul de magistrală asociat ferestrei respective. Ferestrele de adrese predefinite permit folosirea diferitelor modalităţi de lucru, fără a exista suprapuneri şi având numărul limitat la numărul de BUSCONx.

Portul P1 va furniza adresa din interiorul segmentului, chiar dacă BUSCONx setează o magistrală demultiplexată. Aceasta permite folosirea unui decodificator extern conectat la portul P1 pentru toate tipurile de magistrală.

Folosirea seturilor BUSCON/ADDRSEL este controlată prin intermediul adreselor rezultate. Când este iniţiat un acces (instrucţiuni sau date), adresa fizică generată defineşte, dacă accesul este făcut intern, folosirea unei ferestre definite de ADDRSEL4...1 sau folosirea configuraţiei implicite din BUSCON0. După iniţializarea registrelor active, sunt selectate şi evaluate automat prin interpretarea adresei fizice. Nu sunt necesare comutări sau selecţii suplimentare pe durata execuţiei programului, cu excepţia utilizării unui număr mai mare de ferestre faţă de cel implicit.

Atenţie! Nu trebuie schimbată niciodată configurarea zonei de memorie care asigură instrucţiunile curente. Datorită stivei de instrucţiuni, este dificil de determinat care instrucţiune va folosi noua configurare.

Un caz particular de comutare a tipului de magistrală constă în trecerea la magistrală demultiplexată la magistrală multiplexată (figura 2.9.c).

Ciclul de magistrală porneşte prin activarea ALE şi generarea adresei pe porturile P4 şi P1. Întrucât în modul multiplexat este necesară prezenţa adresei şi pe portul P0, aceasta va fi furnizată cu o întârziere de un ciclu maşină care are ca efect întârzierea ciclului de magistrală multiplexată şi măreşte durata semnalului ALE. Timpul suplimentar este necesar pentru ca dispozitivul exterior selectat anterior pe magistrala demultiplexată să elibereze magistrala de date.

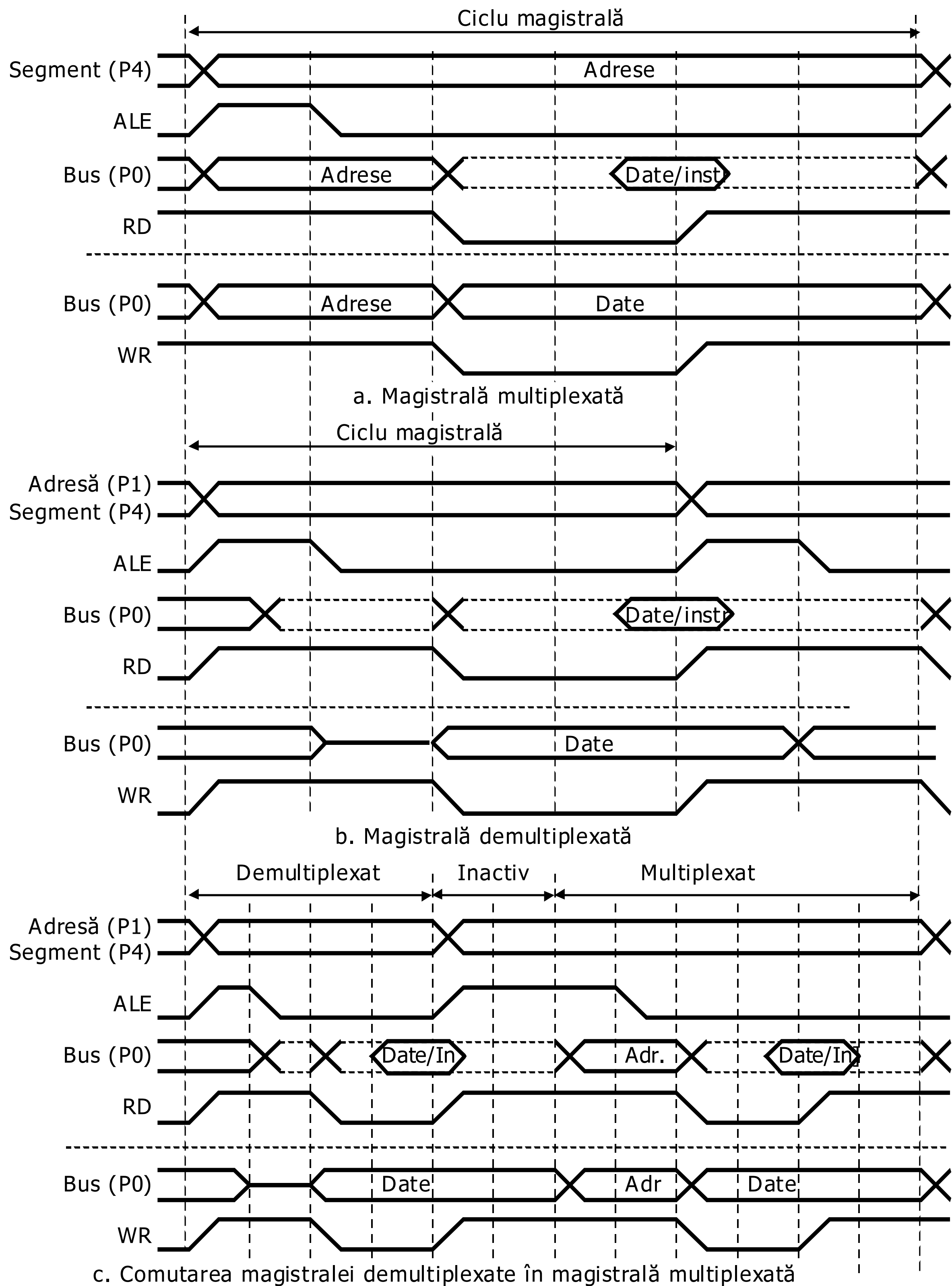


Figura 1.23. Modurile de lucru ale magistralei

d) Dimensiunea magistralei externe de date

EBC poate lucra atât cu periferice de 8 biţi, cât şi cu periferice de 16 biţi. O magistrală de 16 biţi foloseşte portul P0, în timp ce magistrala de 8 biţi foloseşte P0L. Această procedură elimină registrele, bufferele şi decodificatoarele suplimentare, conducând şi la o scădere a timpului de execuţie.

De asemenea, EBC poate controla accesul pe magistrala de 8 biţi a cuvintelor (16 biţi), respectiv acces pe magistrală de 16 biţi a octeţilor:

- pentru accesul pe magistrală de 8 biţi, cuvintele sunt împărţite în două accese succesive. Primul este accesat octetul inferior, apoi cel superior. Asamblarea celor doi octeţi este transparentă pentru utilizator, ea intrând în sarcina EBC;
- accesul unui octet pe o magistrală de 16 biţi necesită utilizarea semnalelor $\overline{BHE}$ (*byte high enable*) pentru octetul superior, respectiv A0 pentru octetul inferior. Astfel, cei doi octeţi ai memoriei pot fi validaţi independent fiecare faţă de celălalt sau împreună, pentru accesul cuvintelor.

Pentru scrierea unui octet la un periferic extern care are o singură intrare de selecţie $\overline{CS}$ şi două validări pentru scriere $\overline{WR}$ pentru cei doi octeţi, EBC poate genera direct aceste două semnale, eliminând circuitul extern pentru combinarea semnalului $\overline{WR}$ cu A0 şi ALE. În acest caz $\overline{WR}$ serveşte ca $\overline{WRL}$ (*write low byte*) iar $\overline{BHE}$ ca $\overline{WRH}$ (*write high byte*). Acest mod de lucru pentru $\overline{WR}$ şi $\overline{BHE}$ este setat de bitul $\overline{WRCFG}$ din registrul SYSCON.

Citirea unui octet pe o magistrală de 16 biţi se face prin accesarea de către 80C167 a întregului cuvânt, ulterior înlăturând octetul de prisos. Totuşi, trebuie avută grijă în situaţia citirii dispozitivelor care îşi schimbă starea în timpul citirii, de exemplu registre FIFO, registre de întreruperi etc. În acest caz octeţii trebuiesc citiţi folosind semnalele $\overline{BHE}$ şi A0.

Caracteristicile sistemului în situaţia utilizării celor patru combinaţii de tipuri de magistrală şi lăţimi a magistralei de date sunt prezentate în tabelul 2.10.

Tabelul 1.23

Mod magistrală	Rată transfer (8/16/32 biţi)	Necesităţi sistem	Linii IO libere
8 biţi, multiplexată	1.5/3/6	Registru, buffer 8 biţi	P1
8 biţi, demultiplexată	1/2/4	Buffer 8 biţi	P0H
16 biţi, multiplexată	1.5/1.5/3	Registru, buffer 16 biţi	P1
16 biţi, demultiplexată	1/1/2	–	–

e) Generarea segmentului de adrese

Pe durata adresărilor externe, EBC generează un număr programabil de linii de adrese pe portul P4, pentru a extinde adresele de pe porturile P0 şi/sau P1, aceasta conducând la creşterea spaţiului adresat.

Numărul de linii pentru adresarea segmentului este ales la iniţializare şi este codificat în câmpul de biţi SALSEL din registrul RP0H. Semnificaţia biţilor SALSEL este prezentată în tabelul 2.11.

Tabelul 1.24

SALSEL	Linii adresă segment	Spaţiu memorie direct adresabil
00	A19...A16	1 MB
01	–	64 kB (minim)
10	A23...A16	16 MB (maxim)

11	A17...A16	256 kB (implicit)

f) Generarea semnalelor de selecţie

Pe durata adresărilor externe, EBC generează un număr de linii pentru selecţia circuitelor externe $\overline{CS}$ (*chip select*) pe portul P6 care permit selectarea directă a perifericelor externe sau a bancurilor de memorie, fără a mai necesita un decodificator extern.

Numărul de linii $\overline{CS}$ este ales la iniţializare şi este codificat în câmpul de biţi CSSEL din registrul RP0H. Semnificaţia biţilor CSSEL este prezentată în tabelul 2.12.

Tabelul 1.25

CSSEL	Linii $\overline{CS}$	Observaţii
00	$\overline{CS2}$...$\overline{CS0}$	
01	$\overline{CS1}$...$\overline{CS0}$	
10	–	Pinii portul P6 liberi pentru IO
11	$\overline{CS4}$...$\overline{CS0}$	Implicit

Ieşirile $\overline{CSx}$ sunt asociate cu registrele BUSCONx şi devin active (nivel 0 LOGIC) pentru orice acces în zona definită de ADDRSELx. Pentru orice acces în afara ferestrei respective, $\overline{CSx}$ devine inactiv (nivel 1 LOGIC).

Semnalele $\overline{CS}$ operează în patru moduri definite de biţii CSWENx şi CSRENx din registrele BUSCON1...4, ca în tabelul 2.13.

Tabelul 1.26

CSWEN	CSREN	Mod de lucru $\overline{CS}$
0	0	$\overline{CS}$ selectare adresă (mod implicit pentru $\overline{CS0}$) Semnalele rămân active pe toată durata ciclului EBC. Un semnal $\overline{CS}$ pentru adresă devine activ sincron cu frontul căzător al semnalului ALE şi devine inactiv la frontul crescător al ALE.
0	1	$\overline{CS}$ numai la citire
1	0	$\overline{CS}$ numai la scriere
1	1	$\overline{CS}$ la citire/scriere Aceste semnale rămân active cât timp semnalele de control asociate ($\overline{RD}$/$\overline{WR}$, $\overline{WRL}$, $\overline{WRH}$) sunt active. Acest mod include şi o întârziere programabilă a semnalelor $\overline{RD}$/$\overline{WR}$.

În concluzie, EBC suportă multe configuraţii pentru circuitele externe. Prin creşterea numărului liniilor de adresă segment, 80C167 poate adresa liniar un spaţiu de 256 kB, 1 MB sau 16 MB. Acest lucru permite folosirea unui număr mare de circuite de memorie, permiţând, de asemenea şi adresarea unui mare număr de periferice, fiind obligatorie utilizarea unor decodificatoare externe.

Liniile $\overline{CS}$ ale circuitului permit adresarea directă a dispozitivelor externe, fără a mai fi necesare decodificatoare.

Aceste facilităţi permit creşterea performanţelor sistemului. De exemplu, folosind 4 linii de adresă segment şi 5 linii $\overline{CS}$, se pot adresa 5 bancuri de memorie de 1 MB, fără nici un circuit extern.

1.15.2 Caracteristici programabile ale magistralei

Importante caracteristici de sincronizare ale EBC pot fi setate de programator pentru a simplifica adaptarea la o mare varietate de dispozitive şi configuraţii de memorii externe.

Se pot programa următorii parametri ai magistralei externe:

- lungimea semnalului ALE;
- 1…15 stări de aşteptare;
- intervalul în care perifericele sunt în înaltă impedanţă;
- timpul de întârziere a semnalelor de citire/scriere;
- controlul semnalului $\overline{\text{READY}}$.

a) Controlul semnalului ALE

Lungimea semnalului ALE şi timpul de menţinere a adresei pe magistrală după frontul căzător al acestuia este controlat de bitul ALECTL din registrele BUSCON. Dacă bitul ALECTL este setat, ciclurile de acces ale ferestrei adresate de BUSCON vor avea semnalul ALE prelungit cu jumătate dintr-o perioadă a ceasului sistem (25 ns pentru frecvenţă de ceas de 20 MHz). De asemenea timpul de menţinere a adresei pe magistrală după frontul căzător al ALE (pe o magistrală multiplexată) va fi prelungită cu jumătate dintr-o perioadă a ceasului sistem. Această procedură este utilizată pentru a asigura un timp suplimentar pentru separarea de pe magistrală a adresei.

Diagrama de timp a acestui semnal este prezentată în figura 2.10.

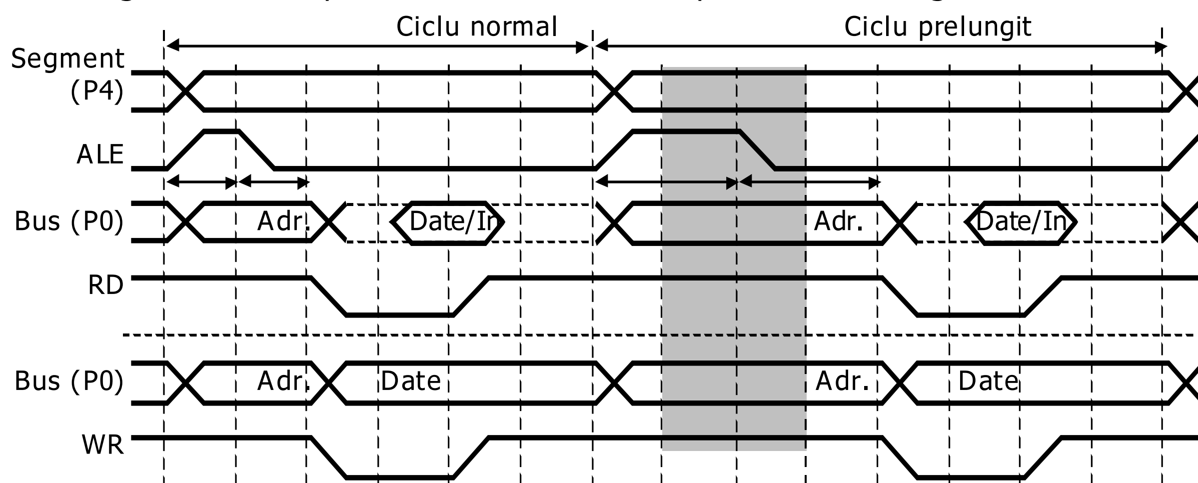


Figura 1.24. Controlul lungimii ALE

b) Stări de aşteptare

Circuitul 80C167 permite ajustarea ciclurilor EBC pentru a se încadra în cerinţele cerute de timpul de acces mai mare al dispozitivelor externe. Aceste stări de aşteptare semnifică timpul total cu care este întârziată apariţia pe magistrală a datelor faţă de adrese. Întârzierea poate fi programată de câmpul de biţi MCTC din BUSCONx în 0…15 incremente de 50 ns (pentru frecvenţa ceasului sistem de 20 MHz).

Dacă accesul extern este necesar instrucţiunii curente, procesorul va aştepta într-o stare inactivă.

Diagrama de timp a procedurii este descrisă în figura 2.11.

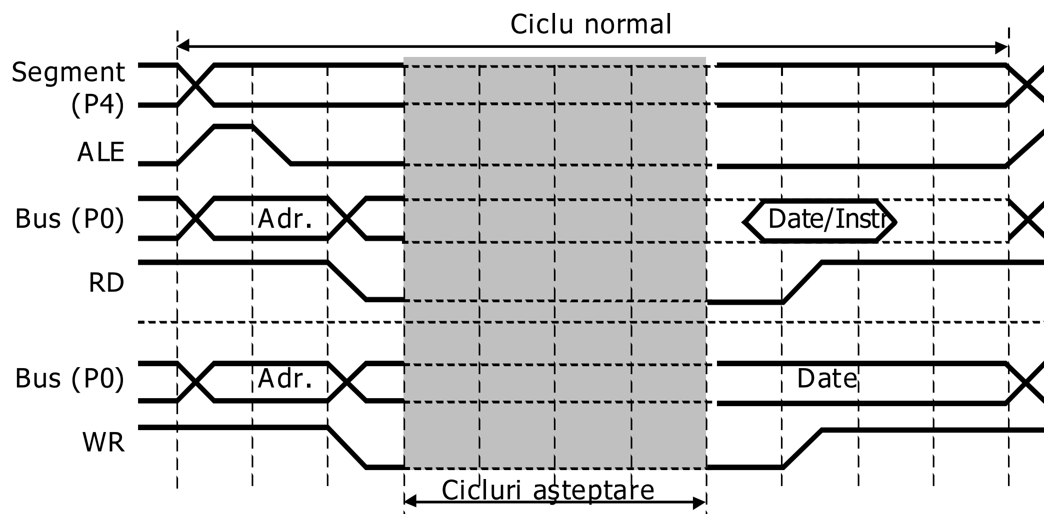


Figura 1.25. Inserare stări de aşteptare

c) Programarea intervalului de înaltă impedanţă

Timpul de înaltă impedanţă este definit ca intervalul după care dispozitivul extern (periferic, memorie etc.) eliberează magistrala de date după dezactivarea semnalului $\overline{\text{RD}}$, conform figurii 2.12.

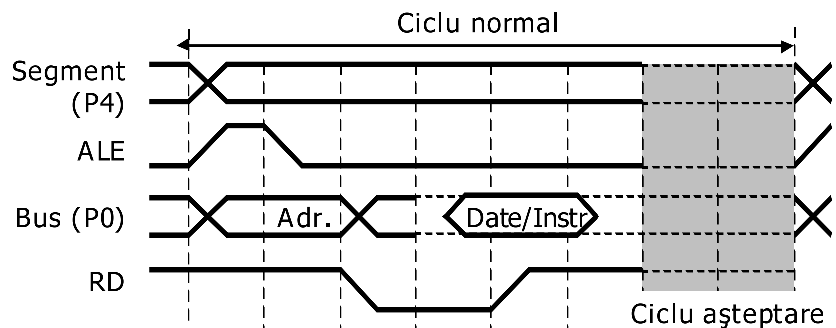


Figura 1.26. Inserare stări de aşteptare

Această procedură este necesară pentru dispozitivele externe care îşi comută prea lent starea magistralei lor de date, din ieşire în înaltă impedanţă.

Pe timpul acestei întârzieri, unitatea centrală nu este inactivă.

Această întârziere este setată funcţie de bitul MTTC al registrului BUSCONx.

d) Întârzierea semnalelor $\overline{\text{RD}}$ şi $\overline{\text{WR}}$

Programatorul are posibilitatea ajustării comenzilor de scriere şi citire pentru a ţine cont de necesităţile dispozitivelor externe. Controlul întârzierii semnalelor $\overline{\text{RD}}$ şi $\overline{\text{WR}}$ stabileşte intervalul între frontul căzător al ALE şi frontul căzător al semnalului respectiv de comandă. Cu această întârziere validată, diferenţa între cele două fronturi căzătoare este de 25 ns (pentru frecvenţa ceasului sistem de 20 MHz). Procedura este descrisă în figura 2.13.

Întârzierea semnalelor $\overline{\text{RD}}$/$\overline{\text{WR}}$ nu măreşte durata ciclului de acces şi, în general nu încetineşte unitatea centrală. Oricum, pentru magistrala multiplexată, driverul unui circuit extern poate intra în conflict cu adresele

80C167 dacă este folosit un semnal $\overline{RD}$ normal. De aceea, modul multiplexat trebuie programat întotdeauna cu semnale $\overline{RD}/\overline{WR}$ întârziate. Controlul acestei proceduri este realizat de bitul RWDC din registrul BUSCONx.

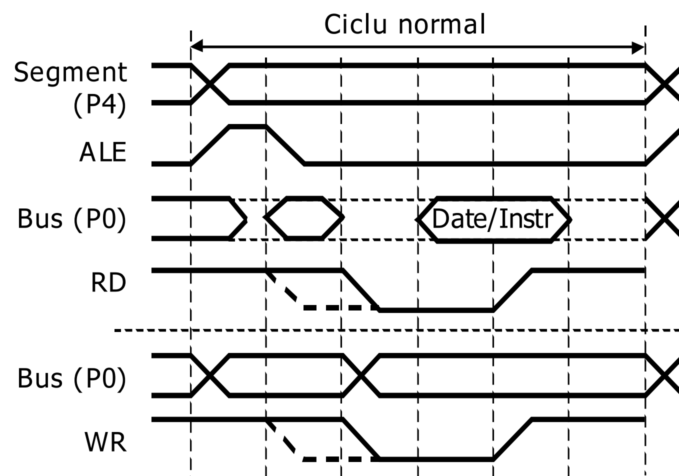


Figura 1.27. Inserare stări de aşteptare semnale $\overline{RD}/\overline{WR}$

e) Controlul semnalului $\overline{READY}$

În situaţii în care stările de aşteptare nu sunt suficiente sau când timpul de răspuns al dispozitivului extern nu este constant, circuitul 80C167 asigură cicluri EBC care sunt terminate (sincron sau asincron) prin intermediului semnalului $\overline{READY}$. În acest caz, unitatea centrală inserează mai întâi un număr de cicluri de aşteptare (0…7) după care intră în mod inactiv, aşteptând ca semnalul $\overline{READY}$ să devină activ.

Funcţionarea în acest mod este programată prin bitul RDYEN din registrul BUSCONx. În acest caz, din câmpul de biţi MCTC utilizat pentru definirea numărului de cicluri de aşteptare au următoarea destinaţie:

- MCTC.2…0 – stabilesc numărul de stări de aşteptare;
- MCTC.3 – defineşte modul sincron sau asincron(MCTC.3=1, respectiv 0).

Semnalul $\overline{READY}$ sincron asigură cea mai mare viteză, dar necesită cunoaşterea cu exactitate a caracteristicilor circuitului extern, precum şi validarea semnalului CLKOUT pentru a fi folosit în exterior.

Semnalul $\overline{READY}$ asincron este mai puţin restrictiv, dar necesită stări suplimentare de aşteptare datorită sincronizării interne.

1.15.3 Registrele speciale ale interfeţei cu magistrala externă

Funcţionarea EBC este controlată de un set de registre speciale. Unii factori sunt stabiliţi de registrul SYSCON (prezentat la 1.14.3.a).

Alte proprietăţi ale magistralei, cum ar fi modul de formare a semnalelor de selecţie, folosirea $\overline{READY}$, lungimea ALE, modul multiplexat sau demultiplexat, întârzierea semnalelor $\overline{RD}/\overline{WR}$, timpii de întârziere etc., sunt condiţionate de registrele BUSCON0…4. Patru din aceste registre dispun de un registru de selecţie a adresei ADRSEL1…4 care permit definirea a patru zone de adresă şi caracteristici specifice ale magistralei în aceste ferestre. Pentru

spaţiul de memorie neocupat de aceste patru registre, controlul este asigurat de BUSCON0.

a) Registrele pentru controlul magistralei (BUSCON0…4, ADDRSEL1…4)

Cele patru perechi de registre BUSCON4…1/ADDRSEL4…1 permit definirea a patru zone separate de adrese în spaţiul de memorie al circuitului 80C167. În fiecare din aceste zone, accesul extern poate fi controlat în unul din cele patru moduri de funcţionare ale magistralei.

Fiecare registru ADDRSEL utilizează în fereastra definită parametrii EBC definiţi de registrul BUSCON asociat. În afara ferestrelor definite de ADDRSEL, operează parametrii determinaţi de BUSCON0.

Structura registrelor este prezentată în tabelul 2.14.

Tabelul 1.27

BUSCON0 (FF0Ch)	R	R	–	RDYEN0	–	BUSACT0	ALECTL0	–	BTYP		MTTC0	RWDC0	MCTC0			
BUSCON1 (FF14h)	CSWEN1	CSREN1	–	RDYEN1	–	BUSACT1	ALECTL1	–	BTYP		MTTC1	RWDC1	MCTC1			
BUSCON2 (FF16h)	CSWEN2	CSREN2	–	RDYEN2	–	BUSACT2	ALECTL2	–	BTYP		MTTC2	RWDC2	MCTC2			
BUSCON3 (FF18h)	CSWEN3	CSREN3	–	RDYEN3	–	BUSACT3	ALECTL3	–	BTYP		MTTC3	RWDC3	MCTC3			
BUSCON4 (FF1Ah)	CSWEN4	CSREN4	–	RDYEN4	–	BUSACT4	ALECTL4	–	BTYP		MTTC4	RWDC4	MCTC4			

CSWEN	0h: semnalul $\overline{CS}$ este independent de semnalele de scriere ($\overline{WR}$,$\overline{WRL}$,$\overline{WRH}$); 1h: semnalul $\overline{CS}$ este generat pe durata semnalului de scriere.
CSREN	0h: semnalul $\overline{CS}$ este independent de semnalul de citire ($\overline{RD}$); 1h: semnalul $\overline{CS}$ este generat pe durata semnalului de citire.
RDYEN	Validare intrare $\overline{READY}$. Descris la 1.15.2.e)
BUSACT	0h: magistrala externă dezactivată; 1h: magistrala externă în domeniul ADDRSEL corespunzător este activată.
ALECTL	Control lungime semnal ALE. Descris la 1.15.2.a)
BTYP	Configurare magistrală externă. Descris la 1.15.1.
MTTC	Configurare timp întârziere. Descris la 1.15.2.c)
RWDC	Control întârziere semnale $\overline{RD}$/$\overline{WR}$. Descris la 1.15.2.d)
MCTC	Număr cicluri aşteptare. Descris la 1.15.2.b)

ADDRSEL1 (FE18h) ADDRSEL2 (FE1Ah) ADDRSEL3 (FE1Ch) ADDRSEL4 (FE1Eh)	RGSAD												RGSZ			

RGSAD	Setare adresă de start fereastră.
RGSZ	Selectare mărime fereastră.

b) Registrul de control la iniţializare (RP0H)

Acest registru stabileşte numărul de semnale generate de 80C167 utilizate pentru selectarea circuitelor externe, precum şi pentru generarea adresei segmentului.

Registrul RP0H nu poate fi schimbat prin program, dar citirea sa oferă informaţii referitoare la configurarea sistemului.

Structura registrului este prezentată în tabelul 2.15.

Tabelul 1.28

RP0H (F108h)	–	–	–	–	–	–	–	–	X	X	X		CSSEL	–

X	Configurare periferice X-BUS. Rezervaţi pentru periferice X-BUS.
SALSEL	Selectarea liniilor de adresă pentru segment. Descrisă la 1.15.1.e).
CSSEL	Selectarea semnalelor $\overline{CS}$. Descrisă la 1.15.1.f).

Atenţie! EBC este validat cât timp cel puţin unul din biţii BUSACT ai BUSCONx este setat.

Portul P1 va genera adresa în interiorul segmentului cât timp unul din registrele BUSCONx selectează un mod demultiplexat, chiar şi în cazul unor cicluri de magistrală multiplexate.

Ferestrele de adrese definite de ADDRSEL nu se pot suprapune.

Ferestrele de adrese definite de ADDRSEL se pot suprapune cu memoria internă dar în acest caz accesul la memorie nu se va face prin intermediul EBC.

La orice acces în zona internă de memorie, EBC este inactivă.

1.15.4 Starea inactivă a interfeţei cu magistrala externă

În timpul modului inactiv al EBC, magistrala externă este definită după cum urmează:

- portul P0 este în înaltă impedanţă;
- portul P1 (dacă a fost utilizat pentru adrese) conţine ultima adresă folosită;
- portul P4 (numai pinii utilizaţi) păstrează ultima adresă a segmentului folosit;
- portul P6 indică semnalele $\overline{CS}$ corespunzătoare adresei;
- ALE este în 0 LOGIC;
- $\overline{RD}$/$\overline{WR}$ sunt în 1 LOGIC.

1.15.5 Arbitrarea magistralei externe

În sistemele de mare performanţă poate fi obligatorie partajarea resurselor externe, cum ar fi bancurile de memorie, între mai multe controlere. Circuitul 80C167 oferă această facilitate prin posibilitatea arbitrării accesului la magistrala sa externă şi, deci, la dispozitivele sale externe.

Această arbitrare a magistralei, permite unui circuit extern principal să capete controlul EBC prin intermediul semnalului $\overline{HOLD}$. 80C167 acceptă această cerere de magistrală, răspunzând cu semnalul $\overline{HLDA}$ şi trecând liniile magistralei în înaltă impedanţă. Noul circuit principal va putea adresa acum

dispozitivele externe prin intermediul aceloraşi linii ale magistralei EBC. În acest timp, circuitul 80C167 secundar îşi poate continua programul dar fără a mai avea acces la magistrala externă.

Dacă procesorul secundar solicită un acces la magistrala sa externă ocupată de un alt procesor principal, solicită accesul la magistrala proprie prin intermediul semnalului $\overline{BREQ}$.

Arbitrarea magistralei externe este validată prin setarea bitului HLDEN din registrul PSW. Dacă bitul este şters, 80C167 nu va răspunde la cererile $\overline{HOLD}$ sosite de la alte procesoare principale.

a) Cedarea magistralei

Accesul la EBC este cerut prin sosirea semnalului $\overline{HOLD}$. După completarea ciclului curent al EBC (dacă există), eliberează magistrala externă şi acordă accesul prin emiterea semnalului $\overline{HLDA}$. În timpul acestei stări, 80C167 setează magistrala externă după cum urmează:

- liniile de adrese şi date în înaltă impedanţă;
- ALE este ţinut în starea 0 LOGIC;
- liniile de comandă $\overline{RD}$,$\overline{WR}$/$\overline{WRH}$,$\overline{BHE}$/$\overline{WRH}$ sunt ţinute în starea 1 LOGIC;
- liniile CSx sunt în starea 1 LOGIC sau în înaltă impedanţă.

Dacă circuitul necesită acces la magistrala sa externă în acest mod, activează cererea de magistrală $\overline{BREQ}$ pentru a semnaliza circuitului de arbitrare a magistralei. $\overline{BREQ}$ poate fi activat numai pe parcursul acestui mod.

b) Preluarea magistralei

Magistrala externă este *redată* circuitului 80C167 prin trecerea liniei $\overline{HOLD}$ în starea 1 LOGIC.

Funcţie de logica de arbitrare, EBC poate fi returnat în două circumstanţe:

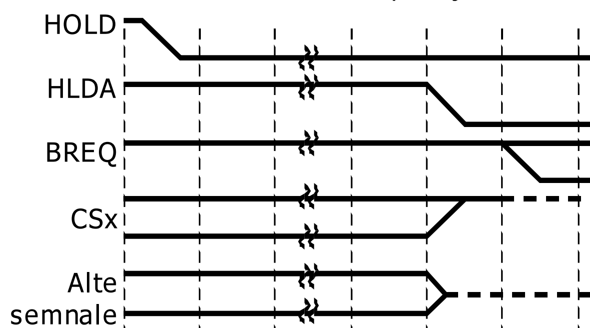
- circuitul extern nu mai solicită resursele partajate ale sistemului şi renunţă *din proprie iniţiativă* la acestea;
- 80C167 are nevoie să acceseze magistrala externă şi o revendică prin intermediul semnalului $\overline{BREQ}$. Logica de arbitrare poate să decidă sau nu dezactivarea semnalului $\overline{HLDA}$ pentru a elibera magistrala, funcţie de prioritatea diferitelor activităţi.

Diagramele de timp a protocolului de arbitrare a magistralei sunt descrise în figura 2.12.a. şi b.

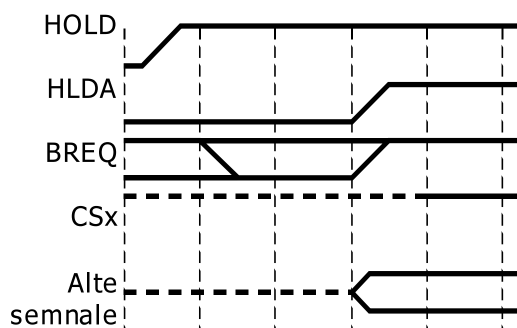
1.15.6 Interfaţa cu magistrala X-BUS

Circuitul 80C167 este înzestrat cu o interfaţă proprie care permite legătura perifericelor interne, speciale sau la comandă, cu unitatea centrală.

În prezent, interfaţa X-BUS este aptă să suporte până la trei periferice de acest tip.



a. Cedarea magistralei



b. Obţinerea magistralei

Figura 1.28. Arbitrarea magistralei externe

Pentru fiecare periferic este rezervată o zonă de memorie controlată de registrele XBCON şi XADRS. Perifericele X-BUS sunt adresate în mod asemănător cu circuitele externe, pe 8 sau 16 biţi, cu sau fără magistrală separată de adrese.

1.16. Sistemul de întreruperi şi registrele PEC

Arhitectura familiei 80C167 suportă mai multe mecanisme pentru un răspuns rapid şi flexibil la cererile de servicii generate de surse variate, interne sau externe.

Aceste mecanisme includ:

- Tratarea normală a întreruperilor – unitatea centrală suspendă execuţia programului curent şi efectuează un salt la o rutină de tratare a întreruperii. Starea curentă a programului (IP, PSW şi, dacă este validată segmentarea, CSP) este salvată în stiva sistem. O structură cu 16 nivele şi 4 grupuri de prioritate oferă utilizatorului posibilitatea să stabilească ordinea în care sunt rezolvate cererile concomitente.
- Tratarea întreruperilor prin intermediul Controlerului de evenimente de la periferice (PEC) – rezidă într-o manieră mai rapidă de a servi întreruperile. Declanşat de o întrerupere normală, PEC realizează transferul unui octet sau cuvânt între două locaţii din segmentul sistem prin intermediul unuia din cele 8 canale PEC disponibile. Avantajul acestei proceduri constă în eliminarea necesităţii salvării stării programului (IP, PSW şi CSP) şi în viteza sa deosebit de ridicată: unitatea centrală opreşte execuţia programului curent numai un ciclu maşină.

- Tratarea excepţiilor (TRAP) – este activată ca răspuns la condiţiile speciale care survin în timpul execuţiei programului. Un caz special este generat de întreruperea externă nemascabilă $\overline{\text{NMI}}$. Excepţiile hardware au întotdeauna cea mai mare prioritate, necesitând reacţia imediată a sistemului. Pot fi generate şi prin program prin intermediul instrucţiunii TRAP care permite generarea unei întreruperi software cu un vector specificat.
- Tratarea întreruperilor externe – circuitul 80C167 permite conectarea la sursele externe de întrerupere prin intermediul unor intrări pentru întreruperi rapide, standard sau nemascabile. Cu excepţia întreruperii nemascabile şi a resetului, toate celelalte intrări sunt funcţii alternative ale porturilor de intrare-ieşire.

1.16.1 Structura sistemului de întreruperi

Circuitul 80C167 asigură 56 de întreruperi care pot fi asignate la 16 nivele de întrerupere.

Pentru a asigura un program modular şi compact, fiecare sursă de întrerupere sau PEC este controlat de un registru de control a întreruperii şi un vector de întrerupere. Registrul de control conţine indicatorii de cerere de întrerupere şi validare a întreruperii, precum şi prioritatea întreruperii. Fiecare cerere este activată de un eveniment specific, funcţie de modul de operare ales. Singurele excepţii constau în întreruperile generate de erorile celor două interfeţe seriale; pentru detectarea tipului de eroare produs, este necesară verificarea registrului de control a interfeţei seriale.

Familia 80C167 asigură un sistem de întreruperi vectorizate, adică pentru fiecare vector de întrerupere, sunt rezervate locaţii în memorie pentru reset, excepţii sau întreruperi. În momentul în care survine o cerere de întrerupere, unitatea centrală execută un salt la adresa specificată de vectorul de întrerupere, permiţând o identificare rapidă a sursei care a generat-o; excepţiile hardware de tip B împart însă, acelaşi vector, pentru identificare fiind necesară analiza registrului TFR (*trap flag register*). Pentru întreruperile software, vectorul de întrerupere este specificat în operandul instrucţiunii (un număr în domeniul 0…3Fh).

Locaţiile vectorilor constituie o tabelă de salturi dispusă la sfârşitul segmentului sistem. Tabela de salturi permite executarea unor salturi oriunde în memorie, la adresa rutinelor de tratare a întreruperilor sau excepţiilor. Intrările în tabela de salturi sunt dispuse la adresele de început ale segmentului 0. Fiecare intrare ocupă două cuvinte, cu excepţia vectorilor excepţiilor hardware şi resetului care ocupă 4 sau 8 cuvinte. Adresa vectorilor de întrerupere rezultă din înmulţirea cu 4 a numărului excepţiei.

Tabelul 2.16 enumeră toate cele 56 de întreruperi sau cereri PEC ale circuitului 80C167, vectorii asociaţi şi adresele lor, precum şi numărul excepţiei. De asemenea, sunt prezentate mnemonicele corespunzătoare ale indicatorilor cerere întrerupere şi validare întrerupere; mnemonicele sunt

compuse dintr-o rădăcină care specifică sursa şi un sufix care specifică destinaţia (IR – cerere de întrerupere, respectiv IE – validare întrerupere).

Ultimele patru locaţii din tabel sunt destinate perifericelor X-BUS. În situaţia în care nu există astfel de periferice conectate, locaţiile respective pot fi folosite ca surse de întreruperi software.

Tabelul 1.29

Sursa întreruperii sau cererii PEC	Indicator cerere	Validare indicator	Vector întrerupere	Adresă vector	Număr excepţie
Registru CAPCOM 0	CC0IR	CC0IE	CC0INT	00’0040h	10h
Registru CAPCOM 1	CC1IR	CC1IE	CC1INT	00’0044h	11h
Registru CAPCOM 2	CC2IR	CC2IE	CC2INT	00’0048h	12h
Registru CAPCOM 3	CC3IR	CC3IE	CC3INT	00’004Ch	13h
Registru CAPCOM 4	CC4IR	CC4IE	CC4INT	00’0050h	14h
Registru CAPCOM 5	CC5IR	CC5IE	CC5INT	00’0054h	15h
Registru CAPCOM 6	CC6IR	CC6IE	CC6INT	00’0058h	16h
Registru CAPCOM 7	CC7IR	CC7IE	CC7INT	00’005Ch	17h
Registru CAPCOM 8	CC8IR	CC8IE	CC8INT	00’0060h	18h
Registru CAPCOM 9	CC9IR	CC9IE	CC9INT	00’0064h	19h
Registru CAPCOM 10	CC10IR	CC10IE	CC10INT	00’0068h	1Ah
Registru CAPCOM 11	CC11IR	CC11IE	CC11INT	00’006Ch	1Bh
Registru CAPCOM 12	CC12IR	CC12IE	CC12INT	00’0070h	1Ch
Registru CAPCOM 13	CC13IR	CC13IE	CC13INT	00’0074h	1Dh
Registru CAPCOM 14	CC14IR	CC14IE	CC14INT	00’0078h	1Eh
Registru CAPCOM 15	CC15IR	CC15IE	CC15INT	00’007Ch	1Fh
Registru CAPCOM 16	CC16IR	CC16IE	CC16INT	00’00C0h	30h
Registru CAPCOM 17	CC17IR	CC17IE	CC17INT	00’00C4h	31h
Registru CAPCOM 18	CC18IR	CC18IE	CC18INT	00’00C8h	32h
Registru CAPCOM 19	CC19IR	CC19IE	CC19INT	00’00CCh	33h
Registru CAPCOM 20	CC20IR	CC20IE	CC20INT	00’00D0h	34h
Registru CAPCOM 21	CC21IR	CC21IE	CC21INT	00’00D4h	35h
Registru CAPCOM 22	CC22IR	CC22IE	CC22INT	00’00D8h	36h
Registru CAPCOM 23	CC23IR	CC23IE	CC23INT	00’00DCh	37h
Registru CAPCOM 24	CC24IR	CC24IE	CC24INT	00’00E0h	38h
Registru CAPCOM 25	CC25IR	CC25IE	CC25INT	00’00E4h	39h
Registru CAPCOM 26	CC26IR	CC26IE	CC26INT	00’00E8h	3Ah
Registru CAPCOM 27	CC27IR	CC27IE	CC27INT	00’00ECh	3Bh
Registru CAPCOM 28	CC28IR	CC28IE	CC28INT	00’00E0h	3Ch
Registru CAPCOM 29	CC29IR	CC29IE	CC29INT	00’0110h	44h
Registru CAPCOM 30	CC30IR	CC30IE	CC30INT	00’0114h	45h
Registru CAPCOM 31	CC31IR	CC31IE	CC31INT	00’0118h	46h
Timer T0	T0IR	T0IE	T0INT	00’0080h	20h
Timer T1	T1IR	T1IE	T1INT	00’0084h	21h
Timer T2	T2IR	T2IE	T2INT	00’0088h	22h
Timer T3	T3IR	T3IE	T3INT	00’008Ch	23h
Timer T4	T4IR	T4IE	T4INT	00’0090h	24h
Timer T5	T5IR	T5IE	T5INT	00’0094h	25h
Timer T6	T6IR	T6IE	T6INT	00’0098h	26h
Timer T7	T7IR	T7IE	T7INT	00’00F4h	3Dh
Timer T8	T8IR	T8IE	T8INT	00’00F8h	3Eh
GPT 2 CAPREL	CRIR	CRIE	CRINT	00’009Ch	27h
ADC completă	ADCIR	ADCIE	ADCINT	00’00A0h	28h
A/D eroare depăşire	ADEIR	ADEIE	ADEINT	00’00A4h	29h
ASC0 emisie	S0TIR	S0TIE	S0TINT	00’00A8h	2Ah
ASC0 buffer emisie	S0TBIR	S0TBIE	S0TBINT	00’011Ch	47h
ASC0 recepţie	S0RIR	S0RIE	S0RINT	00’00ACh	2Bh
ASC0 eroare	S0EIR	S0EIE	S0EINT	00’00B0h	2Ch
SSC emisie	SCTIR	SCTIE	SCTINT	00’00B4h	2Dh

SSC recepţie	SCRIR	SCRIE	SCRINT	00’00B8h	2Eh
SSC eroare	SCEIR	SCEIE	SCEINT	00’00BCh	2Fh
Canal PWM 0...3	PWMIR	PWMIE	PWMINT	00’00FCh	3Fh
Periferic X-BUS 0	XP0IR	XP0IE	XP0INT	00’0010h	40h
Periferic X-BUS 17	XP1IR	XP1IE	XP1INT	00’0104h	41h
Periferic X-BUS 2	XP2IR	XP2IE	XP2INT	00’0108h	42h
Periferic X-BUS 3	XP3IR	XP3IE	XP3INT	00’010Ch	43h

Sistemul de întreruperi este controlat global de registrul PSW (descris în paragraful 1.14.3.b). Suplimentar, fiecare din cele 56 de întreruperi deţin propriul lor registru de control (numele este format dintr-o rădăcină care desemnează sursa întreruperii şi sufixul IC – *interrupt control*).

Structura celor 56 de registre este identică cu cea din tabelul 2.17.

Tabelul 1.30

…IC	–	–	–	–	–	–	–	–	…IR	…IE	ILVL	GLVL
…IR	0h: nici o cere de întrerupere de la modul asociat; 1h: sursa a emis o cerere de întrerupere.											
…IE	0h: cererea de întrerupere invalidată; 1h: cererea de întrerupere validată.											
ILVL	Nivel prioritate – defineşte nivelul priorităţilor pentru arbitrarea cererilor. Are valori de la Fh (cea mai mare prioritate) până la 0h (cea mai scăzută prioritate).											
GLVL	Grup prioritate – folosit pentru a departaja mai multe întreruperi simultane care au aceeaşi prioritate. Cel mai prioritar este grupul 3 iar cel mai puţin grupul 0.											

Indicatorul …IR este setat de hardware în momentul în care survine o cerere de întrerupere de la modul. Indicatorul este şters automat o dată cu intrarea în rutina de tratare, cu excepţia canalelor PEC la care câmpul COUNT a fost decrementat până la 00h. Aceasta declanşează o întrerupere normală ca răspuns la transferul unui bloc complet de date de către PEC.

Câmpurile de biţi ILVL şi GLVL sunt folosite pentru arbitrarea întreruperilor şi, eventual, pentru o arbitrare secundară în situaţia unor cereri cu aceeaşi prioritate sosite simultan.

Atenţie! Toate întreruperile cu aceeaşi prioritate trebuie să fie programate în grupuri cu priorităţi diferite.

Pentru întreruperile servite de PEC, numărul canalului asociat este derivat din câmpurile ILVL şi GLVL. Astfel, programând o sursă cu prioritatea 15, sunt selectate canalele PEC7…4 în timp ce o prioritate 14 selectează canalele PEC3…0 (numărul canalului PEC este determinat de cel mai puţin semnificativ bit al ILVL la care se adaugă cei doi biţi ai GLVL). În acest mod cererile PEC simultane sunt tratate în ordinea priorităţii, de la PEC7 la PEC0.

Tabelul 2.18 prezintă lista excepţiilor hardware şi software, locaţia vectorilor asociaţi, numărul şi priorităţilor excepţiilor.

Întreruperile software pot fi generate de la orice adresă de vector, între 00’0000h şi 00’01FCh. Prioritatea întreruperii software este cea definită de câmpul de biţi ILVL din registrul PSW (descris în paragraful 1.14.3.b).

Tabelul 1.31

	Tip excepţie	Indicator TFR	Vector	Locaţie vector	Număr excepţie	Prioritate excepţie
RESET	Reset hardware Reset software Reset timer watchdog	–	RESET	00'0000h	0h	III
Tip A	Întrerupere nemascabilă Depăşire superioară stivă Depăşire inferioară stivă	NMI STKOF STKUF	NMITRAP STOTRAP STOTRAP	00'0008h 00'0010h 00'0018h	2h 4h 6h	II
Tip B	Cod nedefinit Instrucţiune protejată cu format incorect Acces cuvânt la adresă impară Salt la adresă impară Acces ilegal magistrală externă	UNDOPC PRTFLT ILLOPA ILLINA ILLBUS	BTRAP	00'0028h	0Ah	I
Rezervat		–	–	2Ch…3Ch	0Bh…0Fh	
Excepţii software (TRAP)		–	–	0h…1FCh	00h…7Fh	ILVL (PSW)

1.16.2 Funcţionarea canalelor Controlerului pentru evenimente de la periferice (PEC)

Modulul PEC al circuitului 80C167 furnizează 8 canale pentru mutarea unui octet sau cuvânt între două locaţii din segmentul 0. Acesta este cel mai rapid răspuns posibil la un eveniment şi, de regulă, este suficient pentru tratarea întreruperii de la unele periferice (de exemplu interfeţele seriale, convertorul analog numeric etc.).

Fiecare canal este controlat de un registru de control/numărător, PECCx şi de o pereche de indicatori, sursă şi destinaţie, pentru transferul datelor, SRPCx, respectiv DSTPx (*source pointer*, *destination pointer*).

Structura registrelor PECCx, identice între ele, este prezentată în tabelul 2.19.

Tabelul 1.32

Registru	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PECC0 – FEC0h PECC1 – FEC2h PECC2 – FEC4h PECC3 – FEC6h PECC4 – FEC8h PECC5 – FECAh PECC6 – FECCh PECC7 – FECEh	–	–	–	–	–	INC		BWT	COUNT							

INC	'00': '01': '10': '11':	indicatorii nu sunt modificaţi; incrementează DSTPx cu 1 sau 2 (funcţie de BWT); incrementează SRCPx cu 1 sau 2 (funcţie de BWT); rezervat.
BWT	'0': '1':	transfer 16 biţi; transfer 8 biţi.

<table>
<tr><td rowspan="7">COUNT</td><td colspan="4">Destinat numărării transferurilor PEC.
Conţinutul câmpului COUNT stabileşte acţiunea canalului PEC: efectuarea unui număr oarecare de deplasări, transfer continuu sau nici un transfer.</td></tr>
<tr><td>COUNT anterior</td><td>COUNT modificat</td><td>Indicator întrerupere</td><td>Acţiunea canalului PEC</td></tr>
<tr><td>FFh</td><td>FFh</td><td>0</td><td>Transfer continuu.</td></tr>
<tr><td>FEh…02h</td><td>FDh…01h</td><td>0</td><td>Numărul de transferuri stabilit de COUNT.</td></tr>
<tr><td>01h</td><td>00h</td><td>1</td><td>Oprire servicii PEC; generare întrerupere.</td></tr>
<tr><td>00h</td><td>00h</td><td>(1)</td><td>Nici o acţiune.</td></tr>
</table>

<table>
<tr><td>DSTP0 – 00’FCE2h
DSTP1 – 00’FCE6h
DSTP2 – 00’FCEAh
DSTP3 – 00’FCEEh</td><td>DSTP4 – 00’FCF2h
DSTP5 – 00’FCF6h
DSTP6 – 00’FCFAh
DSTP7 – 00’FCFEh</td><td>SRCP0 – 00’FCE0h
SRCP1 – 00’FCE4h
SRCP2 – 00’FCE8h
SRCP3 – 00’FCECh</td><td>SRCP4 – 00’FCF0h
SRCP5 – 00’FCF4h
SRCP6 – 00’FCF8h
SRCP7 – 00’FCFCh</td></tr>
<tr><td colspan="4">SRPCx şi DSTPx specifică locaţiile între care datele vor fi transferate.</td></tr>
</table>

În mod normal, canalul PEC permite servirea unui număr specificat de cereri până când COUNT, prin decrementare, ajunge la valoarea 00h. În această situaţie este activată o întrerupere specifică numărului canalului PEC.

Transferul continuu este selectat dacă valoarea COUNT este iniţializată cu FFh. În acest caz COUNT nu este decrementat şi canalul PEC respectiv va servi orice cerere până când va fi dezactivat.

Când COUNT este decrementat de la 01h la 00h după un transfer, indicatorul de întrerupere nu este şters, generând o nouă cerere de întrerupere de la aceeaşi sursă.

Dacă valoarea lui COUNT este 00h, canalul respectiv este inactiv, în schimb este activată rutina de tratare asociată evenimentului. Aceasta permite alegerea, dacă o întrerupere de nivel 15 sau 14 este tratată de PEC sau de rutina de tratare.

Transferurile PEC sunt efectuate numai dacă prioritatea lor este mai mare decât prioritatea procesorului. Toate sursele de cereri de întrerupere trebuie să folosească fiecare canale PEC diferite, altminteri pentru cereri simultane va fi efectuat un singur transfer.

Indicatorii sursă şi destinaţie (SRCPx, respectiv DSTPx) desemnează locaţiile între care datele sunt mutate. Fiecare pereche de indicatori este asociată câte unuia din cele 8 canale PEC. Transferul datelor prin canalele PEC nu foloseşte registrele DPP întrucât SRCPx şi DSTPx sunt folosite numai în interiorul segmentului zero.

Locaţiile indicatorilor pentru canalele PEC neutilizate pot fi folosite pentru păstrarea datelor, ca memorie RAM.

Atenţie! Setarea transferului unui cuvânt (BWT=0) obligă respectivul canal PEC să lucreze cu indicatorii sursă şi destinaţie la adrese pare.

1.16.3 Priorităţile sistemului de întreruperi

Întreruperile propriu-zise şi transferurile PEC pot fi validate, arbitrate şi, eventual, dacă au *câştigat* arbitrajul, pot fi servite sau dimpotrivă, pot fi dezactivate, situaţie în care cererea este neglijată şi nu este servită.

Validarea şi invalidarea întreruperilor poate fi făcută prin trei mecanisme:

- Biţii de control (…IE) permit comutarea fiecărei surse de cereri, astfel încât modulul respectiv poate emite o cerere de întrerupere sau nu. De asemenea, există şi posibilitatea validării/invalidării globale prin bitul IEN din registrul PSW;
- Nivelul de prioritate selectează automat un grup de întreruperi care vor fi recunoscute, neglijând celelalte surse. Prioritatea unei surse care câştigă arbitrarea este comparată în permanenţă cu prioritatea unităţii centrale (biţii ILVL din PSW), întreruperea fiind servită numai dacă are o prioritate mai mare ca a unităţii centrale. Un modul intern care are setat nivelul de prioritate 0 va avea dezactivată întreruperea.
- Instrucţiunile ATOMIC şi EXTend dezactivează automat toate cererile de întrerupere pe durata următoarelor 1…4 instrucţiuni.

Administrarea sistemului de întreruperi se face prin crearea unor clase de întreruperi, clase care acoperă un set de întreruperi cu aceeaşi importanţă; întreruperile din aceeaşi clasă nu trebuie să incomodeze una pe alta. Circuitul 80C167 realizează aceasta prin două procedee:

- Clase cu până la 4 membri care folosesc acelaşi nivel de întrerupere (ILVL) dar sunt diferenţiate prin grupul de priorităţi (GLVL). Aceasta este funcţionarea implicită a sistemului de întreruperi;
- Se pot realiza clase cu mai mult de 4 membri prin asocierea a două nivele de priorităţi (ILVL), fiecare cu grupurile sale. Fiecare rutină de tratare a întreruperilor din interiorul acestei clase va seta nivelul unităţii centrale la un cel mai mare nivel de prioritate din clasă. Toate cererile cu prioritate egală sau mai mică vor fi omise, adică nici o întrerupere a clasei definite nu va fi servită.

1.16.4 Salvarea stării programului pe durata întreruperii

Înainte ca o întrerupere să fie servită, starea programului curent este salvată în stiva sistem. Automat, sunt salvate, în ordine, registrele PSW, CSP (dacă este validată segmentarea) şi IP.

Nivelul curent al priorităţii procesorului este adus la valoarea priorităţii întreruperii care este servită. Dacă este în curs de execuţie o înmulţire sau împărţire este setat bitul MULIP din registrul PSW. Indicatorul …IR al întreruperii servite este şters. Registrul IP este încărcat cu vectorul asociat întreruperii iar CSP este şters (dacă segmentarea este validată). Registrele DPP şi CP nu sunt afectate.

În momentul executării instrucţiunii de întoarcere din rutina de tratare a întreruperii (RETI), informaţiile din stivă sunt descărcate în ordine inversă: IP, CSP şi PSW.

Programatorul, de regulă, trebuie să salveze în rutina de tratare toate registrele folosite. Normal, aceste registre sunt salvate în stivă la începerea rutinei şi readuse din stivă înainte de comanda de revenire în programul principal.

Circuitul 80C167 permite ca printr-o singură instrucţiune, SCXT, să salveze toate registrele de lucru (GPR). Instrucţiunea nu salvează în stivă cele 16 registre ci, pur şi simplu, modifică registrul CP care conţine adresa de bază a bancului GPR. Totuşi, celelalte registre utilizate eventual în subrutină (DPPx, MDH, MDL etc.) trebuie salvate clasic, în stiva sistem.

1.16.5 Timpul de răspuns la întrerupere

Timpul de răspuns la întrerupere reprezintă timpul între setarea unui indicator de cerere a întreruperii şi momentul în care unitatea centrală dă controlul rutinei de tratare a întreruperii.

Datorită modului de lucru al unităţii centrale prin stiva de instrucţiuni, înainte de a se da controlul rutinei de tratare, sunt executate instrucţiunile prezente în stivă; în concluzie, timpul de execuţie al acestora influenţează timpul de achitare al întreruperii.

Timpul minim de răspuns este de 5 tacte (125 ns pentru procesor la 40 MHz) şi este atins cu respectarea următoarelor condiţii: instrucţiunile sunt citite din memoria ROM internă, nu se execută accesări ale memoriei externe şi setarea indicatorului de întrerupere s-a produs în ultima fază a unui ciclu de instrucţiune. Dacă indicatorul de întrerupere este setat pe prima fază a unei instrucţiuni, timpul de răspuns este de 6 tacte (150 ns pentru procesor la 40 MHz).

Timpul de răspuns creşte corespunzător pentru orice întârziere produsă de instrucţiunea curentă (N) sau cele două anterioare (N-1 şi N-2) executate înainte de intrarea în rutină, cum ar fi:

- dacă instrucţiunea N modifică registrul PSW şi instrucţiunea N-1 a actualizat indicatorii de stare, timpul de răspuns poate creşte cu două tacte (50 ns);
- dacă instrucţiunea N citeşte un operand din memoria ROM internă, sau dacă este o instrucţiune de apel sau revenire din subrutină, excepţie software sau acces de tipul `MOV Rn,[Rm+#Data16]` , timpul de răspuns poate creşte cu două tacte (50 ns);
- condiţiile interne între instrucţiunile N-2/N-1, N-1/N sau N impun o modificare a registrelor PSW sau SP, timpul de răspuns poate creşte cu un tact (25 ns).

Cazul cel mai defavorabil este atins pentru 12 tacte (300 ns pentru procesor la 40 MHz). În general, programatorul trebuie să evite următoarele:

- încărcarea instrucţiunilor din locaţii externe;
- citirea operanzilor din locaţii externe;
- scrierea rezultatului în locaţii externe.

Problema este diferită în cazul folosirii canalelor PEC, modalitate care, prin definiţie, este mai rapidă.

Timpul minim de răspuns este de 3 tacte (75 ns pentru procesor la 40 MHz) şi este atins cu respectarea următoarelor condiţii: instrucţiunile sunt

citite din memoria ROM internă, nu se execută accesări ale memoriei externe şi setarea indicatorului de întrerupere s-a produs în ultima fază a unui ciclu de instrucţiune. Dacă indicatorul de întrerupere este setat pe prima fază a unei instrucţiuni, timpul de răspuns este de 4 tacte (100 ns pentru procesor la 40 MHz).

În mod similar cu întreruperile standard, funcţie de condiţiile specifice întâlnite în momentul setării indicatorului de întrerupere, timpul de răspuns pentru servirea PEC poate creşte, dar nu mai mult de 9 tacte (225 ns pentru procesor la 40 MHz).

1.16.6 Întreruperile externe

Cu toate că circuitul 80C167 nu are disponibili pini special dedicaţi achiziţionării întreruperilor externe, există mai multe posibilităţi de a reacţiona la evenimente externe asincrone folosind un număr de linii de intrare-ieşire ca intrări de întreruperi.

Semnalele externe pot fi conectate la:

- Pinii CC0IO…CC31IO (intrări comparare/ieşiri captură) de la modulele CAPCOM;
- Pinii T4IN, T2IN – intrări timer;
- CAPIN – intrarea captură a bancului de timere GPT2.

Pentru fiecare din aceşti pini, declanşarea întreruperii sau transferului PEC pot fi produse de tranziţii ale semnalului de intrare fie pozitive, fie negative, fie ambele.

Selectarea frontului este realizată într-un registru de control al perifericului asociat portului respectiv. Prioritatea întreruperii este determinată de registrul de control al întreruperii de la modulul respectiv iar vectorul rutinei de tratare va fi cel prestabilit pentru modul.

Atenţie! Pentru a putea fi folosit ca intrare de întrerupere externă, pinii trebuie setaţi ca intrări în registrul de control al portului respectiv.

În tabelul 2.20 sunt prezentaţi pinii porturilor care pot fi folosiţi ca surse de întreruperi externe, funcţiile de bază a pinilor şi registrele de control.

Tabelul 1.33

Pin	Funcţie de bază	Registru de control
P2.0…15	Registru 0…15 CAPCOM	CC0…CC15
P8.0…7	Registru 16…23 CAPCOM	CC16…CC23
P1H4…7	Registru 24…27 CAPCOM	CC24…CC27
P7.4…7	Registru 28…31 CAPCOM	CC28…CC31
P3.2	Intrare timer auxiliar T2	T2CON
P3.5	Intrare timer auxiliar T4	T4CON
P3.7	Intrare captură banc timere GPT2	T5CON

Când un pin CCxIO se foloseşte ca intrare de întrerupere externă, câmpul de biţi CCMODx din registrul corespunzător CCx trebuie setat corespunzător:

- dacă CCMODx=01h, întreruperea este generată de un front crescător pe pinul CCxIO;

- dacă CCMODx=02h, întreruperea este generată de un front descrescător pe pinul CCxIO;
- dacă CCMODx=03h, întreruperea este generată atât de un front crescător, cât şi descrescător pe pinul CCxIO.

În aceste trei cazuri conţinutul timerului CAPCOM va fi păstrat în registrul de captură CCx, indiferent dacă timerul funcţionează sau nu. Dacă indicatorul CCxIE este setat, este solicitat un transfer PEC sau o întrerupere cu vectorul CCxINT.

Pinii T2IN şi T4IN pot fi utilizaţi pentru generarea unei întreruperi externe când timerele T2 sau T4 asociate sunt configurate în modul capturare (câmpurile T2M sau T4M din T2CON, respectiv T4CON, sunt egale cu 03h).

Frontul activ al semnalului este determinat de câmpurile T2IR şi T4IR din registrele T2IC, respectiv T4IC:

- front crescător, pentru T2IR sau T4IR egali cu 01h;
- front crescător, pentru T2IR sau T4IR egali cu 02h;
- ambele fronturi, pentru T2IR sau T4IR egali cu 02h.

În aceste trei cazuri, conţinutul registrului timerului auxiliar T3 va fi capturat în registrele T2 sau T4, funcţie de tranziţiile de pe pinii T2IN sau T4IN. Dacă indicatorii T2IE sau T4IE sunt setaţi, este solicitat un transfer PEC sau o întrerupere cu vectorii T2INT, respectiv T4INT.

Pinul CAPIN diferă puţin de ceilalţi pini de intrare de timer, el putând fi utilizat pentru generarea unei întreruperi fără a afecta funcţiile perifericului. Dacă indicatorul T5SC din registrul T5CON este şters, funcţia de captură a registrului CAPREL nu este activă iar orice tranziţie a semnalului pe pinul CAPIN va seta indicatorul de întrerupere CRIR din registrul CRIC.

Astfel, registrul CAPREL poate fi încă utilizat pentru reîncărcarea timerului T5 din bancul GPT2, în timp ce pinul CAPIN este folosit ca sursă externă de întrerupere.

Câmpul CI din registrul T5CON stabileşte modul de activare a întreruperii funcţie de tranziţia semnalului:

- CI=01h⇒întreruperea este generată de frontul crescător;
- CI=02h⇒întreruperea este generată de frontul descrescător;
- CI=03h⇒întreruperea este generată de ambele fronturi.

În toate situaţiile, dacă bitul CRIE este setat, vor fi solicitate un transfer PEC sau o întrerupere cu vectorul CRINT.

Pinii de intrare descrişi până acum ca surse de întreruperi externe sunt testaţi la fiecare 200 ns de unitatea centrală, astfel încât evenimentele externe sunt explorate şi detectate la 200 ns (pentru procesor la 40 MHz).

Circuitul 80C167 dispune de alţi 8 pini care pot fi utilizaţi ca surse de întrerupere externă, cu diferenţa că aceştia sunt exploraţi la fiecare 25 ns, chiar mai repede decât întreruperile interne.

Este vorba de 8 pini ai portului P2 (P2.8…P2.15, funcţii alternative CC8IO…CC15IO) care pot fi programaţi individual în acest mod de *întrerupere rapidă*, de asemenea, putând selecta tipul tranziţiei semnalului. Registrul de control al întreruperilor externe EXICON este prezentat în tabelul 2.21.

Tabelul 1.34

EXICON (F1C0)	EXI7E	EXI6E	EXI5E	EXI4E	EXI3E	EXI2E	EXI1E	EXI0E
EXIxE	'00': întreruperea externă inactivă; mod implicit; '01': întrerupere pe front crescător; '10': întrerupere pe front descrescător; '11': . întrerupere pe ambele fronturi.							

Toate aceste întreruperi externe folosesc canalele CC8…CC15 şi vectorii lor de întrerupere. Utilizarea pinilor respectivi pentru captură/comparare nu mai este posibilă, dar se pot folosi în continuare ca pini de intrare-ieşire.

Atenţie! Chiar dacă întreruperile de pe aceşti pini sunt eşantionate la 25 ns, arbitrarea şi prelucrarea întreruperilor este făcută tot la 100 ns.

1.16.7 Excepţii

Excepţiile sunt tratate ca întreruperi standard. Totuşi, excepţiile oferă posibilitatea ocolirii procesului de arbitrare a priorităţii, procedură în care este necesară o reacţie imediată a sistemului. Excepţiile sunt nemascabile şi sunt întotdeauna prioritare faţă de întreruperile normale, indiferent de prioritatea acestora. Circuitul 80C167 oferă două astfel de mecanisme:

- excepţii hardware – declanşate de evenimente care apar în timpul execuţiei programului (acces ilegal la memorie, coduri inexistente etc.);
- excepţii software – iniţiate prin program.

a) Excepţiile software

Sunt iniţiate de instrucţiunea TRAP care produce un apel prin program la o rutină de tratare a întreruperii. Numărul excepţiei specificat ca operand al instrucţiunii TRAP defineşte vectorul, de unde va fi executat saltul.

Execuţia instrucţiunii TRAP produce un efect similar cu o întrerupere tratată de acelaşi vector cu deosebirea că nu este afectat nici un indicator de întrerupere.

b) Excepţiile hardware

Excepţiile hardware sunt produse de erori sau stări specifice ale sistemului care survin pe durata rulării unui program şi care nu pot fi detectate în fazele anterioare de proiectare a aplicaţiei. O excepţie poate fi generată şi intenţionat, de exemplu, pentru excepţia UNDOPC (cod inexistent) se pot emula instrucţiuni adiţionale.

În momentul în care a fost detectată una din cele opt excepţii posibile pentru circuitul 80C167, unitatea centrală execută un salt la locaţia vectorizată pentru locaţia respectivă. Funcţie de excepţie, instrucţiunea care a cauzat-o poate fi terminată sau ignorată înainte de a se da controlul rutinei de tratare a excepţiei.

Excepţiile hardware sunt nemascabile şi au întotdeauna prioritate faţă de orice altă activitate a unităţii centrale. Dacă în cursul aceluiaşi ciclu sunt detectate mai multe excepţii, va fi servită excepţia cea mai prioritară (priorităţile excepţiilor sunt prezentate în tabelul 2.18).

În mod automat sunt salvate în stivă registrele PSW, CSP (numai în mod segmentat) şi IP; nivelul întreruperii unităţii centrale din registrul PSW este setat la maxim (15), dezactivând celelalte eventuale întreruperi.

Pentru a restaura starea unităţii centrale, o rutină de tratare a unei excepţii trebuie încheiată la fel ca o rutină de întrerupere normală prin instrucţiunea RETI.

Cele opt excepţii hardware sunt împărţite în două clase după cum urmează:

- Clasa A (prioritate II):
  - întrerupere externă nemascabilă ($\overline{\text{NMI}}$);
  - depăşire inferioară stivă sistem (STKUF);
  - depăşire superioară stivă sistem (STKOF);
- Clasa B (prioritate III):
  - cod instrucţiune nedefinit (UNDOPC);
  - instrucţiune protejată cu format incorect (PRTFLT);
  - acces cuvânt la adresă impară (ILLOPA);
  - salt la adresă impară (ILLINA);
  - acces ilegal la magistrala externă (ILLBUS).

Cele 8 excepţii sunt administrate de 4 vectori de întrerupere (00’0002h–NMI, 00’0010h–STKOF, 00’0018h–STKUF, 00’0028h–toate excepţiile din clasa B), astfel încât pentru departajarea lor este folosit registrul TFR.

Structura registrului TFR (*trap flag register*), precum şi semnificaţia fiecărui indicator sunt prezentate în tabelul 2.22.

Tabelul 1.35

TFR (FFACh)	NMI	STKOF	STKUF	–	–	–	–	–	UNDOPC	–	–	–	PRTFLT	ILLOPA	ILLINA	ILLBUS
NMI	Indicator setat de o tranziţie descrescătoare pe pinul $\overline{\text{NMI}}$. Registrul IP, salvat automat în stivă, conţine adresa următoarei instrucţiuni.															
STKOF	Setat în momentul în care indicatorul stivei sistem este decrementat la o valoare mai mică decât cea definită în registrul STKOV (descris la 1.14.3.f). Pentru restabilirea sistemului în urma acestei excepţii trebuie verificat spaţiul excedentar rămas în stivă (cel puţin de două ori câte 6 octeţi pentru registrele IP, CSP şi PSW). Altfel, soluţia este executarea unei iniţializări a sistemului (instrucţiunea SRST).															
STKUF	Setat în momentul în care indicatorul stivei sistem este incrementat la o valoare mai mare decât cea definită în registrul STKUN (descris la 1.14.3.f).															
UNDOPC	Setat dacă instrucţiunea curentă, decodificată de unitatea centrală, nu este o instrucţiune validă. Registrul IP salvat în stivă conţine adresa instrucţiunii care a produs excepţia. Rutina de tratare poate fi folosită pentru emularea unor instrucţiuni neimplementate.															

PRTFLT	Setat dacă una din instrucţiunile speciale protejate este executată fără a respecta condiţiile specificate. Instrucţiunile protejate sunt: DISWDT, EINIT, IDLE, PWRDN, SRST şi SRVWDT.
ILLOPA	Setat în momentul în care este încercat un acces (scriere sau citire) de operand pe 16 biţi la o adresă impară.
ILLINA	Setat dacă este executat un salt la o adresă impară.
ILLBUS	Setat în condiţiile în care fără a fi definită magistrala externă, există o solicitare de acces la aceasta (încărcare cod, scriere sau citire operanzi).

Atenţie! Rutinele de tratare a excepţiilor trebuie să şteargă indicatorul din registrul TFR.

Setarea prin program a unui indicator din registrul TFR are acelaşi efect cu acela al unei setări hardware.

Iniţializările sistemului (reset, reset software şi reset timer watchdog) pot fi asimilate unor excepţii cu nivel de prioritate I şi vector la adresa 00'0000h. Aceste excepţii au prioritatea cea mai mare şi întrerup orice altă activitate a procesorului.

Excepţiile din clasa A sunt următoarele ca prioritate. În situaţia în care survin simultan întreruperi de clasa A, este stabilită intern următoarea prioritate: NMI, SKTOF, STKUF.

Excepţiile din clasa B sunt cel mai puţin prioritare. Deoarece partajează acelaşi vector de întrerupere, cele cinci evenimente pot fi ordonate ca prioritate numai prin rutina de tratare a excepţiei.

1.17. Porturile de intrare-ieşire

Circuitul 80C167 dispune de un număr de 111 linii de intrare-ieşire organizate după cum urmează:

- un port de 16 biţi (portul P2);
- opt porturi de 8 biţi (porturile P0 format din P0L şi P0H, P1 format din P1L şi P1H, P4, P6, P7 şi P8);
- un port de 15 biţi (portul P3);
- un port de 16 biţi numai pentru intrări (portul P5).

Toate aceste linii pot fi utilizate ca intrări/ieşiri de uz general controlate prin program sau pot fi folosite de modulele interne ori interfaţa cu magistrala externă.

Toate liniile sunt adresabile la nivel de bit; toate liniile sunt programabile individual ca intrări sau ieşiri (cu excepţia portului P5). Unele porturi (P2, P3, P6, P7 şi P8) pot fi programate individual ca ieşiri push-pull sau cu drenă în gol.

Un set de registre speciale controlează funcţionarea porturilor de intrare-ieşire:

- P0L, P0H, P1L, P1H, P2, P3, P4, P5, P6, P7, P8 – registre de date;
- DP0L, DP0H, DP1L, DP1H, DP2, DP3, DP4, DP6, DP7, DP8 – registre control direcţie semnale;

- ODP2, ODP3, ODP6, ODP7, ODP8 – registre control ieşiri.

Fiecare linie de port are cel puţin o funcţie alternativă de intrare sau ieşire asociată.

Dacă pentru o anumită linie este folosită o funcţie alternativă de ieşire, direcţia acestui pin trebuie stabilită ca ieşire (DPx.y=1) cu excepţia unor semnale care sunt folosite direct după iniţializare şi sunt setate automat. Altminteri pinul rămâne în înaltă impedanţă şi nu afectează funcţia alternativă. Bistabilul respectivei linii trebuie setat, întrucât ieşirea este trecută printr-o poartă ŞI-LOGIC cu linia de ieşire a funcţiei alternative.

Dacă pentru o anumită linie este folosită o funcţie alternativă de intrare, direcţia acestui pin trebuie stabilită ca intrare (DPx.y=0 – implicit după iniţializare). Totuşi, dacă la pinul respectiv nu este conectat nici un dispozitiv extern, acesta se poate defini ca ieşire. În acest caz, funcţia alternativă a pinului citeşte valoarea înscrisă în bistabilul de ieşire al portului. Procedura este utilă pentru testare.

Programatorul este responsabil pentru definirea direcţiei majorităţii liniilor de intrare-ieşire dacă sunt utilizate şi funcţiile alternative. Există totuşi anumite linii care comută automat direcţia semnalelor. Ca exemplu poate fi dat portul P0, utilizat ca magistrală multiplexată de interfaţa EBC, care schimbă direcţia de câteva ori pentru încărcarea unei instrucţiuni.

Toate porturile neutilizate pentru funcţiile alternative sunt disponibile utilizatorului ca linii de intrare-ieşire.

Atenţie! Dacă porturile sunt utilizate ca ieşiri, pentru a preveni tranziţii nedorite, este recomandabilă scrierea mai întâi a valorii urmată de setarea direcţiei portului.

Datorită efectelor stivei de instrucţiuni, instrucţiunile care setează pinii unui anumit port nu trebuie să fie succesive.

1.17.1 Portul P0

Cele două porturi de 8 biţi P0L şi P0H reprezintă jumătatea inferioară, respectiv superioară a portului P0. Fiecare port de 8 biţi poate fi scris (inclusiv printr-un transfer PEC) fără a afecta cealaltă jumătate.

Dacă portul este utilizat ca intrare-ieşire, direcţia fiecărei linii poate fi configurată prin registrele corespunzătoare DP0L şi DP0H.

Structura registrelor de control ale portului P0 sunt indicate în tabelul 2.23.

Tabelul 1.36

P0L (FF00h)	–	–	–	–	–	–	–	–	P0L.7	P0L.6	P0L.5	P0L.4	P0L.3	P0L.2	P0L.1	P0L.0
P0H (FF02h)	–	–	–	–	–	–	–	–	P0H.7	P0H.6	P0H.5	P0H.4	P0H.3	P0H.2	P0H.1	P0H.0
DP0L (F100h)	–	–	–	–	–	–	–	–	DP0L.7	DP0L.6	DP0L.5	DP0L.4	DP0L.3	DP0L.2	DP0L.1	DP0L.0

DP0H (F102h)	–	–	–	–	–	–	–	–	DP0H.7	DP0H.6	DP0H.5	DP0H.4	DP0H.3	DP0H.2	DP0H.1	DP0H.0
P0X.y	Registrul de date al portului P0L sau P0H bitul y.															
DP0X.y	0h: linia P0X.y este intrare; 1h: linia P0X.y este ieşire.															

Funcţiile alternative ale portului P0

Dacă EBC este validată, portul P0 este folosit ca magistrală de date sau magistrală multiplexată date/adrese. Dacă interfaţa externă este demultiplexată pe 8 biţi, portul P0H este disponibil pentru utilizare ca linii de intrare-ieşire.

De asemenea, portul este folosit pentru configurarea automată a sistemului la iniţializare. Astfel, iniţial portul este configurat ca intrare şi fiecare linie este prevăzută cu o rezistenţă internă pentru a asigura citirea unor nivele 1 LOGIC. Utilizatorul, prin intermediul unor rezistenţe externe a căror valoare trebuie stabilită funcţie de specificaţiile circuitului (aceste rezistenţe pot rămâne conectate permanent, fără a-i stânjeni funcţionarea), are posibilitatea de a selecta anumiţi pini care vor avea nivel 0 LOGIC.

La sfârşitul iniţializării, configuraţia selectată va fi scrisă în registrul BUSCON0 iar liniile portului P0H vor fi înregistrate în registrul RP0H. În final, rezistenţele interne sunt deconectate de la linii şi portul P0 comută în modul de funcţionare setat.

În timpul accesării magistralei externe în mod multiplexat, pe portul P0 sunt emise mai întâi adresa din interiorul segmentului curent după care portul este comutat ca intrare şi aşteaptă citirea datelor sau instrucţiunilor care urmează. Pe durata ciclurilor de scriere, P0 generează întâi adresa după care scrie octetul sau cuvântul.

Pe durata ciclurilor externe demultiplexate P0 citeşte instrucţiunile sau datele care sosesc ori generează octeţi sau cuvinte de date.

Stabilirea direcţiei portului în situaţia validării EBC se face automat de către hardware. În această situaţie programul nu trebuie să execute scrieri către acest port.

Structura şi direcţia pinilor portului P0 pentru funcţiile alternative sunt prezentate în figura 2.15.

1.17.2 Portul P1

Cele două porturi de 8 biţi P1L şi P1H reprezintă jumătatea inferioară, respectiv superioară a portului P1. Fiecare port de 8 biţi poate fi scris (inclusiv printr-un transfer PEC) fără a afecta cealaltă jumătate.

Dacă portul este utilizat ca intrare-ieşire, direcţia fiecărei linii poate fi configurată prin registrele corespunzătoare DP1L şi DP1H.

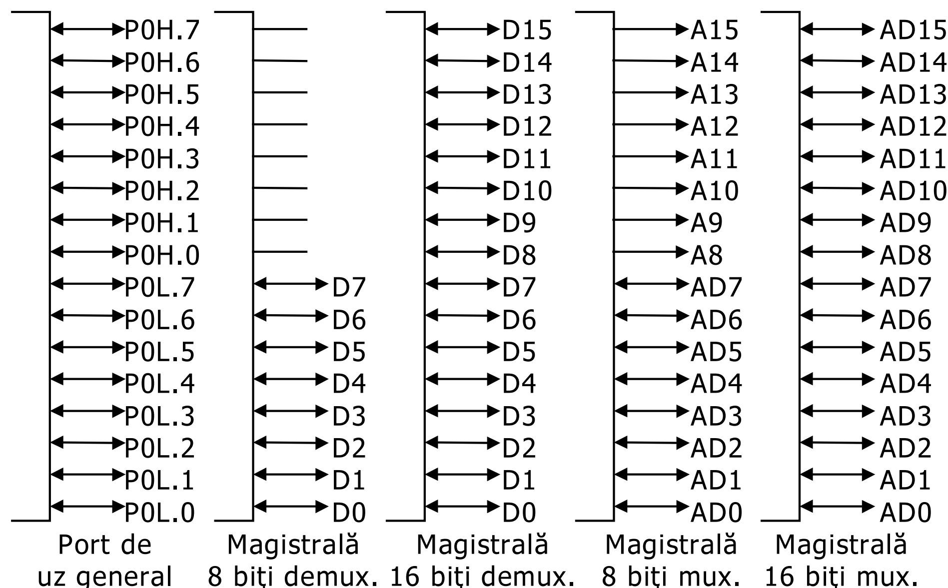


Figura 1.29. Funcţiile alternative ale portului P0

Structura registrelor de control ale portului P1 sunt prezentate în tabelul 2.24.

																Tabelul 1.37
P1L (FF04h)	–	–	–	–	–	–	–	–	P1L.7	P1L.6	P1L.5	P1L.4	P1L.3	P1L.2	P1L.1	P1L.0
P1H (FF06h)	–	–	–	–	–	–	–	–	P1H.7	P1H.6	P1H.5	P1H.4	P1H.3	P1H.2	P1H.1	P1H.0
DP1L (F104h)	–	–	–	–	–	–	–	–	DP1L.7	DP1L.6	DP1L.5	DP1L.4	DP1L.3	DP1L.2	DP1L.1	DP1L.0
DP1H (F106h)	–	–	–	–	–	–	–	–	DP1H.7	DP1H.6	DP1H.5	DP1H.4	DP1H.3	DP1H.2	DP1H.1	DP1H.0
P1X.y	Registrul de date al portului P1L sau P1H bitul y.															
DP1X.y	0h: linia P1X.y este intrare; 1h: linia P1X.y este ieşire.															

Funcţiile alternative ale portului P1

P1 este folosit ca magistrală de adrese A0…A15 în situaţia utilizării unei magistrale externe demultiplexate.

Pinii P1H.7…4 pot fi folosiţi ca intrări de captură pentru modulele CAPCOM. De asemenea, aceste patru linii sunt utilizabile ca intrări pentru întreruperi externe. Ca un efect colateral, posibilitatea capturării unor intrări poate fi folosită şi dacă portul este folosit ca magistrală de adrese. Astfel, unele modificări ale liniilor superioare de adrese pot fi detectate şi declanşa cereri de întrerupere.

Pe durata accesării magistralei externe, portul P1 este folosit numai ca magistrală de adrese. Însă în aceeaşi situaţie, dacă accesul este multiplexat şi

nici un registru BUSCONx nu selectează o magistrală demultiplexată, portul P1 poate fi folosit ca port de uz general.

Structura şi direcţia pinilor portului P1 pentru funcţiile alternative sunt prezentate în figura 2.16.

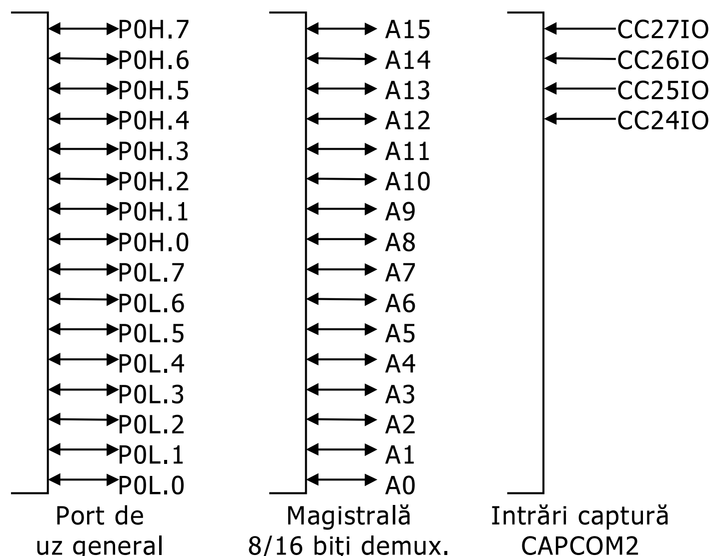


Figura 1.30. Funcţiile alternative ale portului P1

1.17.3 Portul P2

Acest port este utilizat pentru intrări/ieşiri de uz general, pe 16 biţi, direcţia fiecărei linii putând fi selectată din registrul DP2. Fiecare ieşire poate fi comutată în mod push-pull sau drenă în gol prin intermediul registrului ODP2.

Structura registrelor de control ale portului P2 sunt prezentate în tabelul 2.25.

Tabelul 1.38

P2 (FFC0h)	P2.15	P2.14	P2.13	P2.12	P2.11	P2.10	P2.9	P2.8	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
DP2 (FFC2h)	DP2.15	DP2.14	DP2.13	DP2.12	DP2.11	DP2.10	DP2.9	DP2.8	DP2.7	DP2.6	DP2.5	DP2.4	DP2.3	DP2.2	DP2.1	DP2.0
ODP2 (F1C2h)	ODP2.15	ODP2.14	ODP2.13	ODP2.12	ODP2.11	ODP2.10	ODP2.9	ODP2.8	ODP2.7	ODP2.6	ODP2.5	ODP2.4	ODP2.3	ODP2.2	ODP2.1	ODP2.0
P2.y	Registrul de date al portului P2 bitul y.															
DP2.y	0h: linia P2.y este intrare; 1h: linia P2.y este ieşire.															
ODP2.y	0h: linia P2.y este ieşire push-pull; 1h: linia P2.y este ieşire drenă în gol.															

Funcţiile alternative ale portului P2

Toate liniile portului P2 servesc şi ca linii de intrare de captură sau linii de ieşire de comparare pentru modulul CAPCOM1 (CC0IO…CC15IO).

Dacă o linie a portului P2 este folosită ca o intrare de captură, starea bistabilului de intrare, care reprezintă starea pinului, este direcţionată către modulul CAPCOM.

Când o linie a portului P2 este folosită ca o ieşire de comparare (pentru modurile 1 şi 3; detalii suplimentare în paragraful 1.21), îndeplinirea condiţiei afectează direct bistabilul de ieşire al liniei.

În ambele situaţii, utilizatorul are acces liber la pinii portului, chiar dacă sunt folosiţi ca intrări de captură. Dacă programul intenţionează să scrie concomitent cu modulul de comparare, conform regulii generale are prioritate scrierea software.

Toate liniile P2.0…P2.15 pot fi utilizate ca intrări întreruperi externe standard iar P2.8…P2.15 pot fi folosite ca intrări întreruperi externe rapide (EXxIN). De asemenea, P2.15 este întrebuinţat şi ca intrare pentru timerul T7 al modulului CAPCOM2 (T7IN).

Structura şi direcţia pinilor portului P2 pentru funcţiile alternative sunt prezentate în figura 2.17.

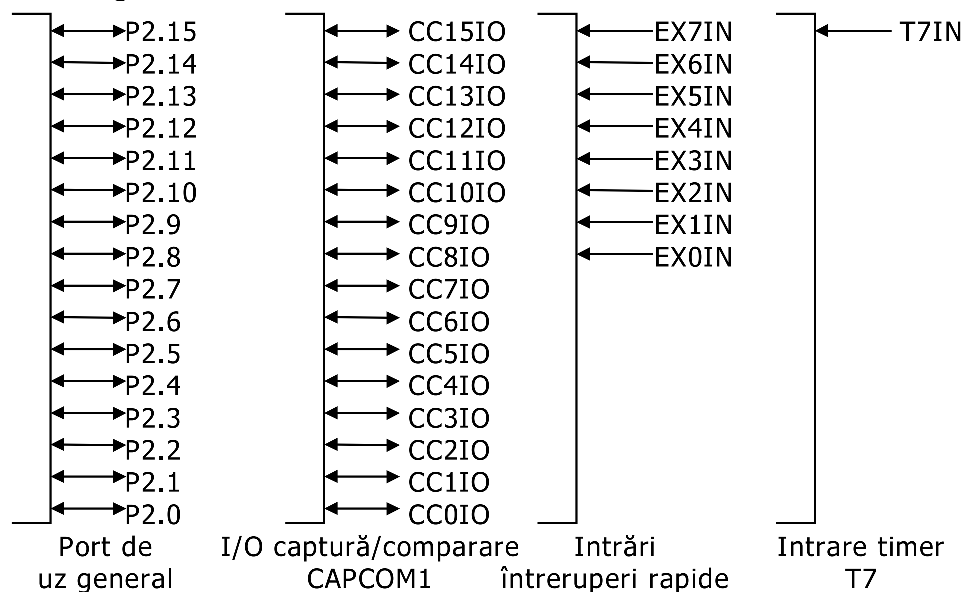


Figura 1.31. Funcţiile alternative ale portului P2

1.17.4 Portul P3

Acest port este utilizat pentru intrări/ieşiri de uz general, pe 15 biţi, direcţia fiecărei linii putând fi selectată din registrul DP3. Majoritatea ieşirilor pot fi comutate în mod push-pull sau drenă în gol prin intermediul registrului ODP3.

Structura registrelor de control ale portului P3 sunt prezentate în tabelul 2.26.

Tabelul 1.39

P3 (FFC4h)	P3.15	–	P3.13	P3.12	P3.11	P3.10	P3.9	P3.8	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
DP3 (FFC6h)	DP3.15	–	DP3.13	DP3.12	DP3.11	DP3.10	DP3.9	DP3.8	DP3.7	DP3.6	DP3.5	DP3.4	DP3.3	DP3.2	DP3.1	DP3.0
ODP3 (F1C6h)	–	–	ODP3.13	–	ODP3.11	ODP3.10	ODP3.9	ODP3.8	ODP3.7	ODP3.6	ODP3.5	ODP3.4	ODP3.3	ODP3.2	ODP3.1	ODP3.0

P3.y	Registrul de date al portului P3 bitul y.
DP3.y	0h: linia P3.y este intrare; 1h: linia P3.y este ieşire.
ODP3.y	0h: linia P3.y este ieşire push-pull; 1h: linia P3.y este ieşire drenă în gol.

Funcţiile alternative ale portului P3

Liniile portului P3 au multiple utilizări care includ intrări/ieşiri de control şi semnale pentru timere, cele două interfeţe seriale, semnalele de sincronizare $\overline{BHE}$/$\overline{WRH}$ şi ieşirea ceasului sistem CLKOUT.

Dacă funcţia alternativă este configurată ca o intrare, citirea se face din bistabilul de intrare care reflectă starea pinului. Aceste funcţii sunt: T0IN, T2IN, T3IN, T4IN (intrări de numărare timere T0, T2, T3 respectiv T4), T3EUD (intrare de numărare sus/jos timer T3), CAPIN (intrare captură GPT2) şi RxD0 (recepţie ASC0).

Când funcţia alternativă este o ieşire, semnalul este trecut printr-o poartă ŞI-LOGIC cu ieşirea portului. Acest lucru implică programatorului setarea liniei ca ieşire (DP3.y=1) şi apoi setarea pinului (P3.y=1). Semnalele de ieşire care folosesc portul P3 sunt: T3OUT (ieşire timer T3), T6OUT (ieşire timer T6), TxD0 (emisie ASC0), $\overline{BHE}$/$\overline{WRH}$ (octet superior valid/scriere octet superior) şi CLKOUT (ceas sistem).

Semnalele MRST (emisie SSC), MTSR (recepţie SSC) şi SCLK (ceas transmisie SSC) sunt semnale atât de intrare cât şi de ieşire.

Structura şi direcţia pinilor portului P3 pentru funcţiile alternative sunt prezentate în figura 2.18.

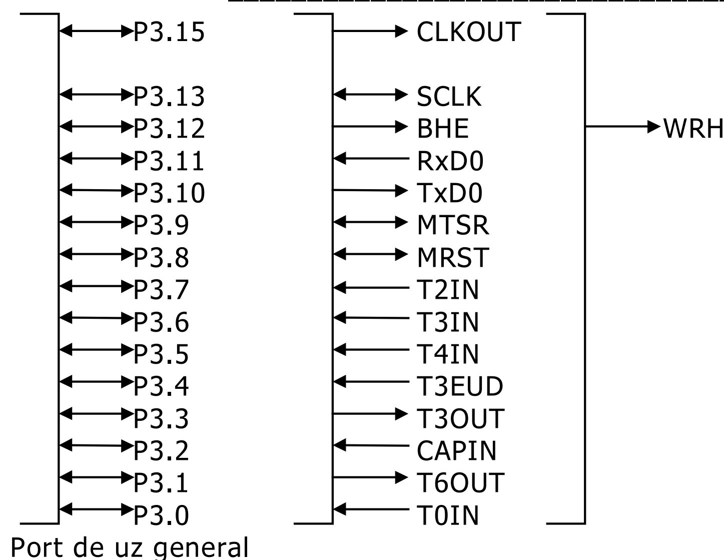


Figura 1.32. Funcţiile alternative ale portului P3

1.17.5 Portul P4

Portul P4 dispune de 8 linii de intrare-ieşire comandate prin registrul P4. Direcţia semnalelor este programată prin registrul DP4. Structura celor două registre este prezentată în tabelul 2.27.

																Tabelul 1.40
P4 (FFC8h)	–	–	–	–	–	–	–	–	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
DP4 (FFCAh)	–	–	–	–	–	–	–	–	DP3.7	DP3.6	DP3.5	DP3.4	DP3.3	DP3.2	DP3.1	DP3.0
P4.y	Registrul de date al portului P4 bitul y.															
DP4.y	0h: linia P4.y este intrare; 1h: linia P4.y este ieşire.															

Funcţiile alternative ale portului P4

Pentru ciclurile externe care folosesc segmentarea memoriei, un număr variabil de pinii ai portului P4 (funcţie de conţinutul câmpului SALSEL din registrul RP0H – descris în paragraful 1.15.3.b) sunt folosiţi pentru generarea adreselor superioare. Eventualii pini rămaşi liberi pot fi întrebuinţaţi ca linii de intrare-ieşire de uz general. Structura şi direcţia pinilor portului P4 pentru funcţiile alternative sunt prezentate în figura 2.19.

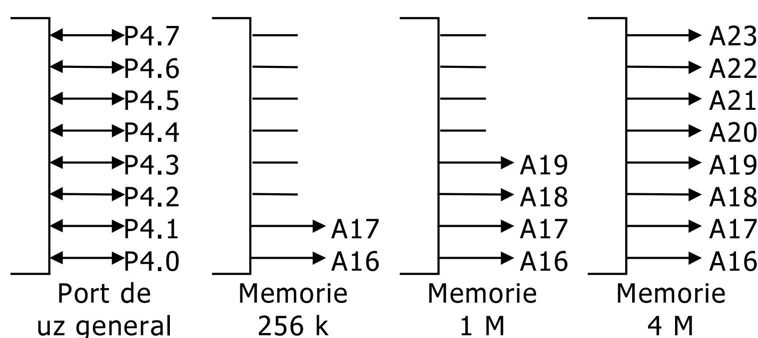


Figura 1.33. Funcţiile alternative ale portului P4

1.17.6 Portul P5

Acest port dispune de 16 linii numai de intrare. Datele scrise în acest port sunt pierdute. Modulul nu are bistabile de ieşire şi nici registru de direcţie. Structura celor registrului de date este prezentată în tabelul 2.28.

Tabelul 1.41

P5 (FFA2h)	–	–	–	–	–	–	–	–	P4.7	P4.6	P4.5	P4.4	P4.3	P4.2	P4.1	P4.0
P5.y	Registru de date al portului P5 bit y (numai citire).															

Funcţiile alternative ale portului P5

Fiecare linie a portului P5 este utilizată şi ca intrare a multiplexorului analogic pentru convertorul analog/numeric. Suplimentar, portul P5 poate asigura şase intrări pentru timere. Structura şi direcţia pinilor portului P5 pentru funcţiile alternative sunt prezentate în figura 2.20.

Figura 1.34. Funcţiile alternative ale portului P5

1.17.7 Portul P6

Acest port este utilizat pentru intrări/ieşiri de uz general, pe 8 biţi, direcţia fiecărei linii putând fi selectată din registrul DP6. Fiecare ieşire poate fi comutată în mod push-pull sau drenă în gol prin intermediul registrului ODP6. Structura registrelor de control ale portului P6 sunt prezentate în tabelul 2.29.

Tabelul 1.42

P6 (FFCCh)	–	–	–	–	–	–	–	–	P6.7	P6.6	P6.5	P6.4	P6.3	P6.2	P6.1	P6.0
DP6 (FFCEh)	–	–	–	–	–	–	–	–	DP6.7	DP6.6	DP6.5	DP6.4	DP6.3	DP6.2	DP6.1	DP6.0
ODP6 (F1CEh)	–	–	–	–	–	–	–	–	ODP6.7	ODP6.6	ODP6.5	ODP6.4	ODP6.3	ODP6.2	ODP6.1	ODP6.0
P6.y	Registrul de date al portului P6 bitul y.															
DP6.y	0h: linia P6.y este intrare; 1h: linia P6.y este ieşire.															

ODP6.y	0h: linia P6.y este ieşire push-pull; 1h: linia P6.y este ieşire drenă în gol.

Funcţiile alternative ale portului P6

Funcţie de conţinutul registrului RP0H pot folosi ca ieşiri liniile portului P6 până la 5 semnale de selecţie $\overline{CSx}$. De asemenea, portul P6 mai este folosit şi de semnalele pentru arbitrarea magistralei externe ($\overline{BREQ}$, $\overline{HLDA}$ şi $\overline{HOLD}$). Structura şi direcţia pinilor portului P6 pentru funcţiile alternative sunt prezentate în figura 2.21.

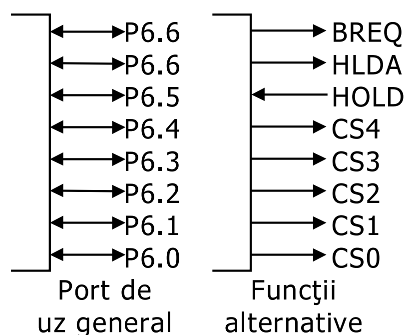


Figura 1.35. Funcţiile alternative ale portului P6

Pentru a asigura selecţia circuitelor în timpul iniţializării sau cedării magistralei, liniile de selecţie $\overline{CSx}$ dispun de rezistenţe interne de pull-up. Dacă este programată vreo ieşire în modul drenă în gol, rezistenţele interne nu vor mai fi active.

1.17.8 Portul P7

Acest port este utilizat pentru intrări/ieşiri de uz general, pe 8 biţi, direcţia fiecărei linii putând fi selectată din registrul DP7. Fiecare ieşire poate fi comutată în mod push-pull sau drenă în gol prin intermediul registrului ODP7. Structura registrelor de control ale portului P7 sunt prezentate în tabelul 2.30.

Tabelul 1.43

P7 (FFD0h)	–	–	–	–	–	–	–	–	P7.7	P7.6	P7.5	P7.4	P7.3	P7.2	P7.1	P7.0
DP7 (FFD2h)	–	–	–	–	–	–	–	–	DP7.7	DP7.6	DP7.5	DP7.4	DP7.3	DP7.2	DP7.1	DP7.0
ODP7 (F1D2h)	–	–	–	–	–	–	–	–	ODP7.7	ODP7.6	ODP7.5	ODP7.4	ODP7.3	ODP7.2	ODP7.1	ODP7.0
P7.y	Registrul de date al portului P7 bitul y.															
DP7.y	0h: linia P7.y este intrare; 1h: linia P7.y este ieşire.															
ODP7.y	0h: linia P7.y este ieşire push-pull; 1h: linia P7.y este ieşire drenă în gol.															

Funcţiile alternative ale portului P7

Liniile P7.4…P7.7 pot fi utilizate ca intrări de captură sau ieşiri de comparare CC28IO…CC31IO. La folosirea lor în acest scop trebuie să se ţină cont de precauţiile descrise la portul P2. Similar, ca celelalte intrări/ieşiri CAPCOM, aceste linii pot fi utilizate şi pentru achiziţionarea unor întreruperi externe standard.

Liniile P7.0…P7.3 pot servi ca ieşiri pentru modulul modulator de impulsuri în durată (PWM). Semnalele de ieşire al acestor module sunt trecute printr-o poartă SAU-EXCLUSIV cu bistabilul de ieşire al portului pentru a permite negarea semnalului PWM (dacă P7.0…3=1). Structura şi direcţia pinilor portului P7 pentru funcţiile alternative sunt prezentate în figura 2.22.

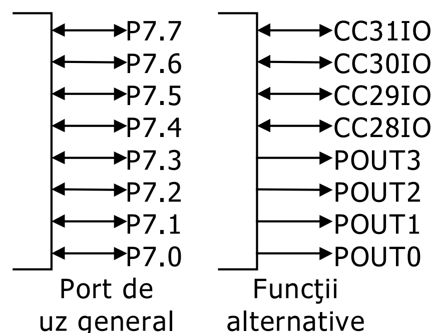


Figura 1.36. Funcţiile alternative ale portului P7

1.17.9 Portul P8

Acest port este utilizat pentru intrări/ieşiri de uz general, pe 8 biţi, direcţia fiecărei linii putând fi selectată din registrul DP8. Fiecare ieşire poate fi comutată în mod push-pull sau drenă în gol prin intermediul registrului ODP8. Structura registrelor de control ale portului P8 sunt prezentate în tabelul 2.31.

Tabelul 1.44

P8 (FFD4h)	–	–	–	–	–	–	–	–	P8.7	P8.6	P8.5	P8.4	P8.3	P8.2	P8.1	P8.0
DP8 (FFD6h)	–	–	–	–	–	–	–	–	DP8.7	DP8.6	DP8.5	DP8.4	DP8.3	DP8.2	DP8.1	DP8.0
ODP8 (F1D6h)	–	–	–	–	–	–	–	–	ODP8.7	ODP8.6	ODP8.5	ODP8.4	ODP8.3	ODP8.2	ODP8.1	ODP8.0
P7.y	Registrul de date al portului P8 bitul y.															
DP8.y	0h: linia P8.y este intrare; 1h: linia P8.y este ieşire.															
ODP8.y	0h: linia P8.y este ieşire push-pull; 1h: linia P8.y este ieşire drenă în gol.															

Funcţiile alternative ale portului P8

Toate liniile portului P8 pot fi folosite, cu precauţiile descrise la portul P2 ca linii de intrare-ieşire pentru modulul CAPCOM şi, de asemenea, pentru achiziţionarea întreruperilor externe. Structura pinilor portului P8 pentru funcţiile alternative sunt prezentate în figura 2.23.

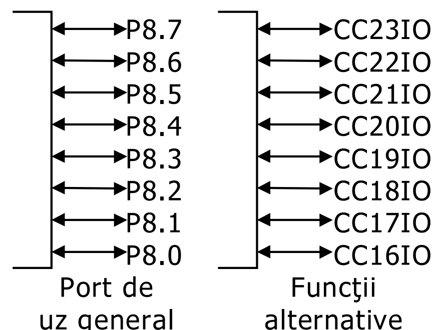


Figura 1.37. Funcţiile alternative ale portului P8

1.18. Modulatorul de impulsuri în durată

Modulul pentru modularea impulsurilor în durată PWM permite generarea a patru semnale modulate în durată independente. Pentru un circuit cu frecvenţa de ceas de 20 MHz, frecvenţa acestor semnale este cuprinsă între 4.8 Hz şi 10 MHz (impulsuri aliniate pe front) sau între 2.4 Hz şi 5 MHz (semnale aliniate central).

Modulul PWM constă în 4 canale independente. Fiecare canal conţine un numărător sus/jos de 16 biţi `PTx`, un registru de 16 biţi pentru perioada de repetiţie `PWx`, un registru de 16 biţi pentru durata impulsului `PPx`, un bistabil virtual, două comparatoare, precum şi logica de control necesară. Lucrul celor patru canale este controlat de două registre `PWMCON0` şi `PWMCON1` iar pentru controlul întreruperilor generate de PWM se foloseşte registrul `PWMIC`.

De asemenea, un control asupra ieşirilor `PWM` îl au şi registrele speciale ale portului `P7` (`P7`, `DP7` şi `ODP7`) prezentate în paragraful 1.17.8.

Schema bloc a modulelor PWM este prezentată în figura 2.24 (zonele haşurate reprezintă registrele aflate sub controlul utilizatorului).

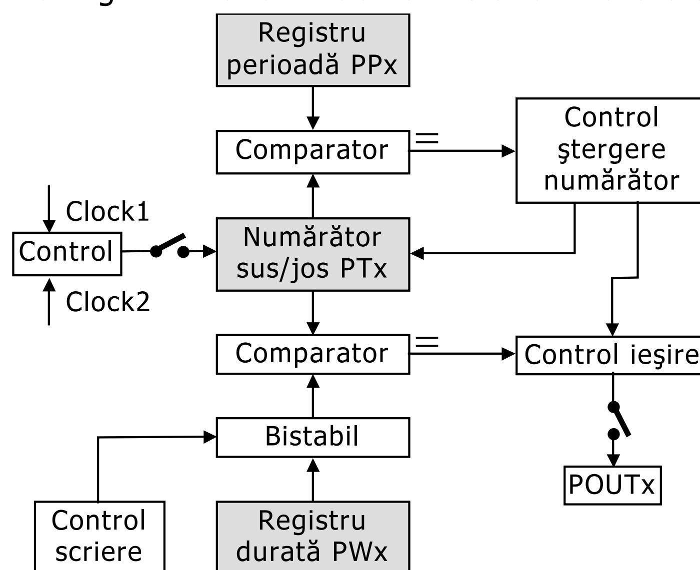


Figura 1.38. Schema bloc a modulului PWM

1.18.1 Moduri de operare

Modulul PWM dispune de patru moduri de funcţionare:

- PWM standard: generează impulsuri aliniate pe front;

- PWM simetric: generează impulsuri aliniate central;
- Salvă (*burst*): generează grupuri de impulsuri;
- Impuls singular (*single shot*): generează un singur impuls.

Atenţie! Ieşirile POUTx sunt trecute printr-o poartă SAU-EXCLUSIV cu ieşirile corespunzătoare ale portului P7. Setarea liniei respective poate produce inversarea semnalului.

a) Modul 0

Modul 0 este selectat prin ştergerea bitului PMx din registrul PWMCON1. În acest mod, numărătorul PTx al canalului respectiv numără crescător până când atinge valoarea registrului de perioadă PPx. Următorul impuls de ceas provoacă iniţializarea numărătorului.

Semnalul de ieşire POUTx este în nivel 1 LOGIC cât timp conţinutul numărătorului este mai mare sau egal cu conţinutul registrului PWx. Semnalul este comutat în nivel 0 LOGIC o dată cu iniţializarea numărătorului.

În concluzie, perioada de repetiţie este dată de relaţia:

$$T_{PWM\ Mod0}=[PPx]+1$$

Factorul de umplere al impulsului generat este programabil între 100% (registrul PWx=0) şi 0% (registrul PWx=PPx).

Acest mod este denumit şi *aliniat pe front* întrucât valoarea registrului PWx afectează numai frontul crescător al impulsului, în timp ce frontul descrescător este determinat de ştergerea numărătorului la atingerea valorii din PPx.

Modul de funcţionare este ilustrat în figura 2.25.a.

b) Modul 1

Modul 1 este ales prin setarea bitului PMx din registrul PWMCON1. În acest mod, numărătorul PTx al canalului respectiv numără crescător până când atinge valoarea registrului de perioadă PPx. Următorul impuls de ceas provoacă schimbarea direcţiei de numărare a numărătorului care continuă să numere descrescător până atinge valoarea 0h. Următorul impuls de ceas comută iar direcţia de numărare, crescător, procedura continuând în acelaşi mod.

Ieşirea PWM este în starea 1 LOGIC cât timp conţinutul PTx este mai mare sau egal cu valoarea PPx. Ieşirea este comutată în 0 LOGIC când valoarea PTx scade sub valoarea PPx.

Perioada semnalului PWM în modul 1 se poate calcula cu relaţia:

$$P_{PWM\ Mod1}=2\cdot[PPx]+2$$

Acest mod este denumit şi *aliniat central* întrucât valoarea registrului PWx afectează fronturile crescătoare şi descrescătoare ale impulsului, în mod simetric.

Modul de funcţionare este ilustrat în figura 2.25.b.

c) Modul 2

Modul 2 este selectat setând indicatorul PB01 din registrul PWMCON1.

Acest mod combină semnalele canalelor PWM0 şi PWM1 (printr-o poartă ŞI-LOGIC) pe pinul de ieşire al canalului PWM0. Semnalul produs de PWM1 este disponibil în continuare pe pinul POUT1.

Modul de funcţionare este ilustrat în figura 2.25.c.

Atenţie! Ieşirile POUTx sunt funcţii alternative ale portului P7. Ieşirile portului P7 pot fi setate să fie cu drenă în gol, situaţie în care, prin intermediul unor rezistenţe externe pot fi realizate conexiuni ŞI-CABLAT între ieşirile POUTx, permiţând suficiente combinaţii între modulele PWM pentru obţinerea unor trenuri de impulsuri.

d) Modul 3

Modul 3 este selectat setând indicatorii PSx din registrul PWMCON1. Acest mod este disponibil numai pentru canalele PWM2 şi 3.

În acest mod, timerul PTx este pornit prin program iar el numără până la atingerea valorii din registrul PPx. Următorul impuls de ceas provoacă ştergerea PTx şi oprirea numărării.

Ieşirea PWM este comutată în nivel 1 LOGIC cât timp PTx≥PWx. Semnalul este comutat în 0 LOGIC după ştergerea PTx, adică PTx<PWx.

În concluzie, setarea modulului PWM în acest mod produce un impuls singular care are frontul crescător declanşabil prin program iar durata controlată prin registrele PWx şi PPx.

Chiar după declanşarea numărării (realizată prin setarea indicatorului PTRx din registrul PWMCON0), durata impulsului poate fi modificată prin program, scriind în registrul PTx. Aceste multiple redeclanşări sunt posibile oricând timerul este activ (PTRx=1). De exemplu, dacă registrul PTx este încărcat cu valoarea din PPx, următorul impuls va declanşa oprirea numărătorului.

Modul de funcţionare este ilustrat în figura 2.25.d.

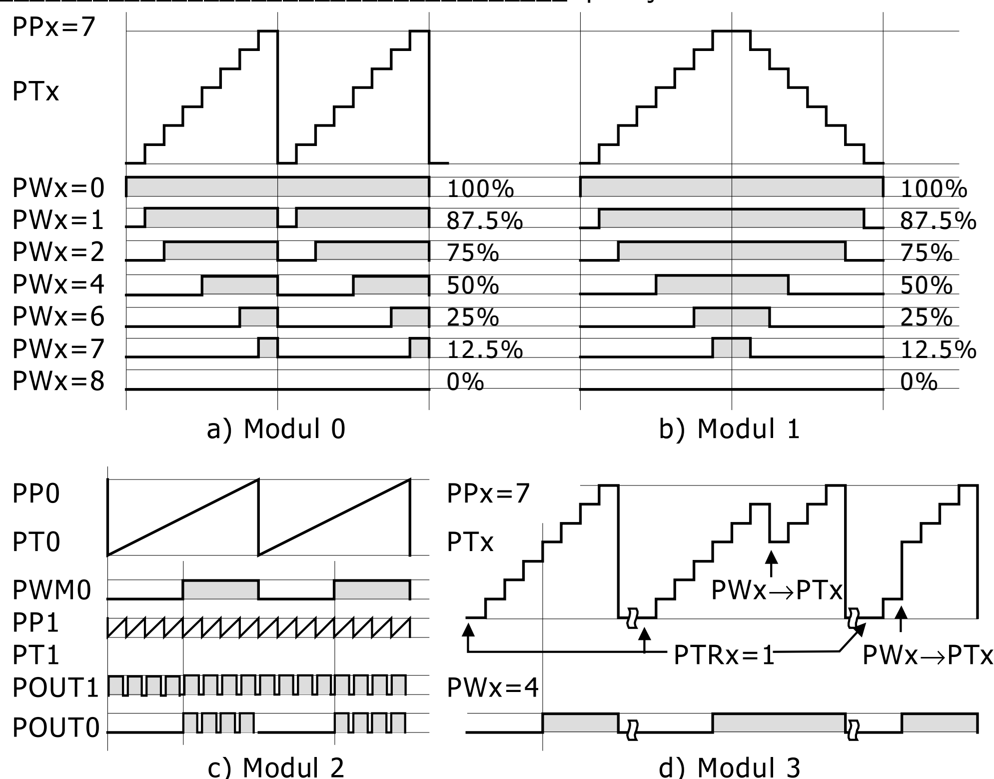


Figura 1.39. Modurile de funcţionare ale PWM

1.18.2 Registrele speciale ale PWM

Modulul PWM este controlat prin intermediul a două seturi de registre:

- formele de undă sunt programate de PTx, PPx şi PWx;
- controlul funcţionării şi a întreruperii este asigurat de registrele PWMCON0, PWMCON1 şi PWMIC.

a) Numărătorul PTx

Prin intermediul bitului PTIx din registrul PWMCON0 se poate selecta pentru fiecare numărător PTx frecvenţa de numărare (fie frecvenţa ceasului sistem, fie aceasta divizată cu 64). Frecvenţa este aplicată numărătoarelor dacă biţii corespunzători PTRx sunt setaţi.

În tabelul 2.32 sunt prezentate câteva frecvenţe produse de PWM pentru un circuit cu ceas sistem la 20 MHz, funcţie de modurile de operare, frecvenţa de intrare şi conţinutul registrului durată impuls.

Tabelul 1.45

Mod	Frecvenţă	PWx 8 biţi	PWx 10 biţi	PWx 12 biţi	PWx 14 biţi	PWx 16 biţi
Mod 0	f_{CPU}	78.13 kHz	19.53 kHz	4.88 kHz	1.22 kHz	305 Hz
	$f_{CPU}/64$	1.22 kHz	305 Hz	76.3 Hz	19.1 Hz	4.77 Hz
Mod 1	f_{CPU}	39.1 kHz	9.77 kHz	2.44 kHz	610 Hz	152.6 Hz
	$f_{CPU}/64$	610 Hz	152.6 Hz	38.15 Hz	9.54 Hz	2.4 Hz

b) Registrul de perioadă PPx

Este un registru de 16 biţi care este folosit pentru programarea perioadei ciclului PWM, adică a frecvenţei de repetiţie. În funcţionare, este comparată

valoarea registrelor PPx şi PTx, la egalitate, funcţie de modul de lucru, executându-se ştergerea registrului PTx şi schimbarea direcţiei de numărare.

c) Registrul de durată PWx

Registrul de 16 biţi PWx controlează valoarea coeficientului de umplere al semnalului PWM.

Unitatea centrală verifică egalitatea între un registru tampon (care păstrează valoarea PWx) şi conţinutul PTx. Registrul tampon este iniţializat la începutul fiecărui ciclu PWM cu conţinutul registrului PWx sau în timpul scrierii acestuia, în ultima situaţie fiind obligatoriu ca numărătorul să fie oprit.

Dacă PTx≥PWx, ieşirea PWM este trecută în starea 1 LOGIC.

Locaţiile registrelor PTx, PPx şi PWx sunt indicate în tabelul 2.33.

Tabelul 1.46

PWM0		PWM1		PWM2		PWM3	
PT0	F030h	PT1	F032h	PT2	F034h	PT3	F036h
PP0	F038h	PP1	F03Ah	PP2	F03Ch	PP3	F03Eh
PW0	FE30h	PW1	FE32h	PW2	FE34h	PW3	FE36h

d) Registrele de control PWMCON0 şi PWMCON1

Registrul PWMCON0 controlează funcţionarea celor patru numărătoare şi administrează întreruperile modulului. Prin intermediul unor instrucţiuni pe câmpuri de biţi (de exemplu BFLDL sau BFLDH) este posibilă comutarea simultană, pentru toate numărătoarele a modului de funcţionare.

Registrul PWMCON1 controlează modul de funcţionare şi semnalele de ieşire a celor patru canale PWM.

Structura celor registre este prezentată în tabelul 2.34.

Tabelul 1.47

PWMCON0 (FF30h)	PIR3	PIR2	PIR1	PIR0	PIE3	PIE2	PIE1	PIE0	PTI3	PTI2	PTI1	PTI0	PTR3	PTR2	PTR1	PTR0
PIRx	0h: nu sunt generate întreruperi; 1h: canalul x generează întrerupere.															
PIEx	0h: întreruperea generată de canalul x dezactivată; 1h: întreruperea generată de canalul x validată.															
PTIx	0h: timerul x numără cu frecvenţa unităţii centrale; 1h: timerul x numără cu $f_{CPU}/64$.															
PTRx	0h: timerul x este deconectat de la intrarea de ceas; 1h: timerul x este conectat de la intrarea de ceas.															
PWMCON1 (FF32h)	PS3	PS2	–	PB01	–	–	–	–	PM3	PM2	PM1	PM0	PEN3	PEN2	PEN1	PEN0
PSx	0h: nu are nici o semnificaţie; 1h: canalul x lucrează în modul 3.															
PB01	0h: nu are însemnătate; 1h: canalele 0 şi 1 lucrează în modul 2.															
PMx	0h: canalul x lucrează în modul 0; 1h: canalul x lucrează în modul 1.															
PENx	0h: ieşirea canalului x dezactivată (se generează numai întrerupere); 1h: ieşirea canalului x funcţională.															

Atenţie! Ştergerea în timpul funcţionării canalului PWMx a bitului PTRx opreşte numărătorul, menţinând neschimbată ieşirea corespunzătoare.

Modificarea prin program a registrului PTx produce actualizarea imediată a ieşirii.

1.18.3 Întreruperile modulului PWM

Fiecare din cele patru canale PWM pot genera o cerere individuală de întrerupere. Fiecare din aceste canale activează un modul de întrerupere PWM care, la rândul său, solicită o întrerupere controlerului de întrerupere.

Rutina de tratare trebuie să determine, pe baza indicatorului PIR din registrul PWMCON0, ce canal a generat întreruperea. Indicatorii PIRx sunt setaţi la începutul unui nou ciclu PWMx.

Atenţie! Indicatorii PIRx nu sunt şterşi automat de circuit la intrarea în rutina de tratare a întreruperii astfel încât este obligatorie ştergerea lor prin program.

Structura registrului PWMIC este identică cu a celorlalte registre de control a întreruperilor descrise în paragraful 1.16.1, tabelul 2.17.

1.19. Convertorul analog numeric

Circuitul 80C167 dispune de un convertor analog/numeric cu o rezoluţie de 10 biţi, un circuit de eşantionare şi un multiplexor analogic pentru selectarea uneia din cele 16 intrări analogice (intrări partajate cu portul P5).

Este recomandabil ca tensiunile de referinţă V_{AREF} şi V_{AGND} să fie generate printr-o sursă separată de cea a circuitelor logice pentru a reduce interferenţele cu alte semnale.

Controlul funcţionării ADC este asigurat de registrele ADCON (selectare mod funcţionare, canal convertit etc.), ADAT şi ADAT2 (rezultate conversie).

Întreruperile generate de convertor sunt administrate de registrele ADCIC şi ADEIC.

Schema bloc a modulului ADC este prezentată în figura 2.26.

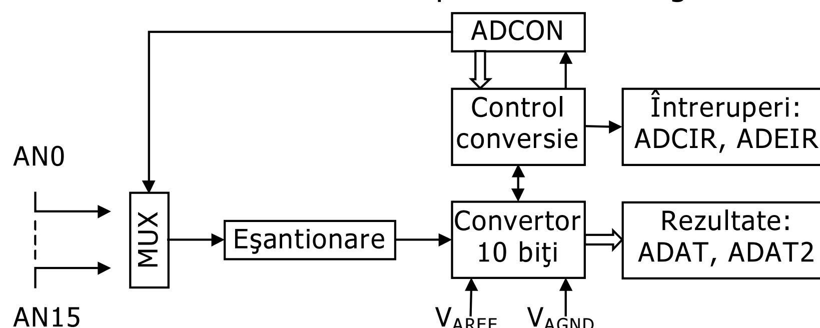


Figura 1.40. Structura internă a modulului ADC

1.19.1 Moduri de lucru

Modulul ADC permite următoarele moduri de conversie:

- conversie singulară;
- conversie continuă;
- conversie multiplă singulară;
- conversie multiplă continuă;

- aşteptare semnal citire rezultat;
- inserare canal.

Funcţionarea modulului ADC este controlată de registrul special ADCON, rezultatul conversiei este păstrat în registrul ADAT iar, în cazul unei conversii inserate, în registrul ADAT2. Structura celor trei registre este prezentată în tabelul 2.35.

Tabelul 1.48

ADCON (FFA0h)	ADCTC		ADSTC		ADCRQ	ADCIN	ADWR	ADBSY	ADST		ADM		ADCH			
ADCTC	Control timp conversie. Descris în paragraful 1.19.2.															
ADSTC	Control timp eşantionare. Descris în paragraful 1.19.2.															
ADCRQ	Indicator cerere inserare canal.															
ADCIN	Validare cerere inserţie canal.															
ADWR	Regim aşteptare semnal citire rezultat.															
ADBSY	Conversie în curs.															
ADST	Start conversie.															
ADM	'00': conversie singulară; '01': conversie continuă; '10': conversie multiplă singulară; '11': conversie multiplă continuă.															
ADCH	Selectare canal intrare. În mod multiplu, selectează primul canal convertit.															
ADAT (FEA0h)	CHNR				–	–	ADRES									
CHNR	Numărul intrării convertite.															
ADRES	Rezultat conversie curentă (10 biţi).															
ADAT2 (F0A0h)	CHNR				–	–	ADRES									
CHNR	Numărul intrării inserate.															
ADRES	Rezultat conversie inserată (10 biţi).															

Conversii singulare

Aceste moduri sunt selectate prin intermediul câmpului ADM având valoarea 0h (conversie singulară) sau 1h (conversie multiplă).

După pornirea convertorului prin intermediul ADST, indicatorul ADBSY va fi setat şi intrarea specificată în ADCH va fi convertită. După terminarea conversiei, indicatorul cerere întrerupere ADCIR va fi setat.

În modul conversie singulară, la sfârşitul operaţiunii curente, convertorul se va opri automat şi va şterge indicatorii ADBSY şi ADST.

În modul conversie multiplă, la sfârşitul operaţiunii curente, convertorul va iniţia automat o nouă conversie a canalului specificat. ADCIR va fi setat la sfârşitul fiecărei conversii.

Dacă bitul ADST este şters prin program în timpul unei conversii, convertorul nu se opreşte decât după ce va finaliza activitatea curentă.

Conversii multiple

Aceste moduri sunt selectate prin programarea câmpului ADM cu valorile 2h (pentru un singur canal) sau 3h (mai multe canale). Acest mod asigură conversia unui şir de intrări analogice, începând cu canalul specificat în câmpul ADCH şi terminând cu canalul 0, fără a fi necesare intervenţii prin program pentru schimbarea numărului canalului.

După pornirea convertorului (bitul ADST setat), canalul specificat în ADCH va fi convertit. După ce conversia a fost finalizată, este setat indicatorul ADCIR iar convertorul porneşte automat o nouă conversie a canalului imediat inferior. ADCIR va fi setat după fiecare conversie completă. După convertirea canalului 0, secvenţa se consideră încheiată.

În modul conversie multiplă singulară, convertorul se va opri automat şi va şterge biţii ADBSY şi ADST.

În modul conversie multiplă continuă, convertorul va iniţia automat o nouă secvenţă de conversii începând cu intrarea specificată de ADCH.

Dacă bitul ADST este şters prin program, convertorul îşi va continua activitatea până la finalul conversiei canalului 0.

Diagrama de timp a unei conversii multiple este prezentată în figura 2.27.a.

Aşteptare semnal citire rezultat

În modul normal de lucru al ADC, dacă un rezultat anterior nu a fost citit din registrul ADAT înainte de terminarea unei noi conversii, rezultatul anterior este pierdut întrucât registrul ADAT va conţine noul rezultat. Pierderea rezultatului anterior este marcată prin setarea indicatorului de depăşire ADEIR.

Este indicată utilizarea acestui mod de lucru pentru a evita generarea unor întreruperi ADEIR şi pierderea unor rezultate, în special în modul continuu.

În acest caz, dacă valoarea anterioară din ADAT nu a fost citită şi rezultatul unei noi conversii este gata, noul rezultat este păstrat într-un registru temporar iar declanşarea unei noi conversii este suspendată. După citirea valorii anterioare din ADAT, registrul temporar este încărcat în registrul ADAT (generând o întrerupere ADCIR) iar conversia suspendată este reluată.

Diagrama de timp a unei conversii cu aşteptarea semnalului de citire este prezentată în figura 2.27.b.

Inserare canal

În acest mod este permisă convertirea unei intrări ADC specifice, chiar dacă modulul execută alte conversii, fără a schimba modul curent de operare. La finalul conversiei *inserate*, ADC îşi continuă activitatea normal.

Atenţie! Câmpul CHNR care determină numărul canalului inserat, nu trebuie modificat în timpul executării unei conversii inserate.

Declanşarea unei conversii inserate poate fi făcută în două moduri:

- setarea prin program a bitului ADCRQ din registrul ADCON;
- o comparare sau captură a registrului CC31.

A doua metodă permite declanşarea inserării unui canal sincronizat cu evenimentele gestionate de registrele CAPCOM (identitatea între registrul timerului şi registrul CAPCOM sau capturarea unui eveniment extern).

Atenţie! Bitul ADCRQ va fi setat de orice întrerupere solicitată de canalul CAPCOM CC31, indiferent dacă modul inserat este activ sau nu. Pentru a preveni orice incidente, este recomandabilă ştergerea bitului ADCRQ înainte de intrarea în modul inserat.

Nu poate fi declanşată o altă conversie inserată dacă alta este deja în curs.

Dacă în timpul inserării unei conversii convertorul este pornit de către program pentru o conversie normală, inserarea este ignorată. Pentru a preveni aceasta, este recomandabilă testarea bitului ADBSY înainte de inserarea unei conversii.

Registrul temporar este folosit pentru păstrarea datelor, atât în modurile normale cât şi în modul de inserare.

Diagrama de timp a unei conversii inserate este prezentată în figura 2.27.c.

1.19.2 Timpii de conversie

Principiul de funcţionare obligă ca, la iniţierea unei conversii, să fie mai întâi încărcat condensatorul din circuitul de eşantionare. Timpul de încărcare al acestui condensator este cunoscut ca *timp de eşantionare*. Convertorul analog/numeric fiind realizat pe principiul registrului cu aproximaţii succesive, necesită 10 paşi, câte unul pentru fiecare bit, pentru finalizarea conversiei. Pe durata acestor 10 paşi, condensatorul de eşantionare este în permanenţă încărcat şi descărcat prin pinul V_{AREF}.

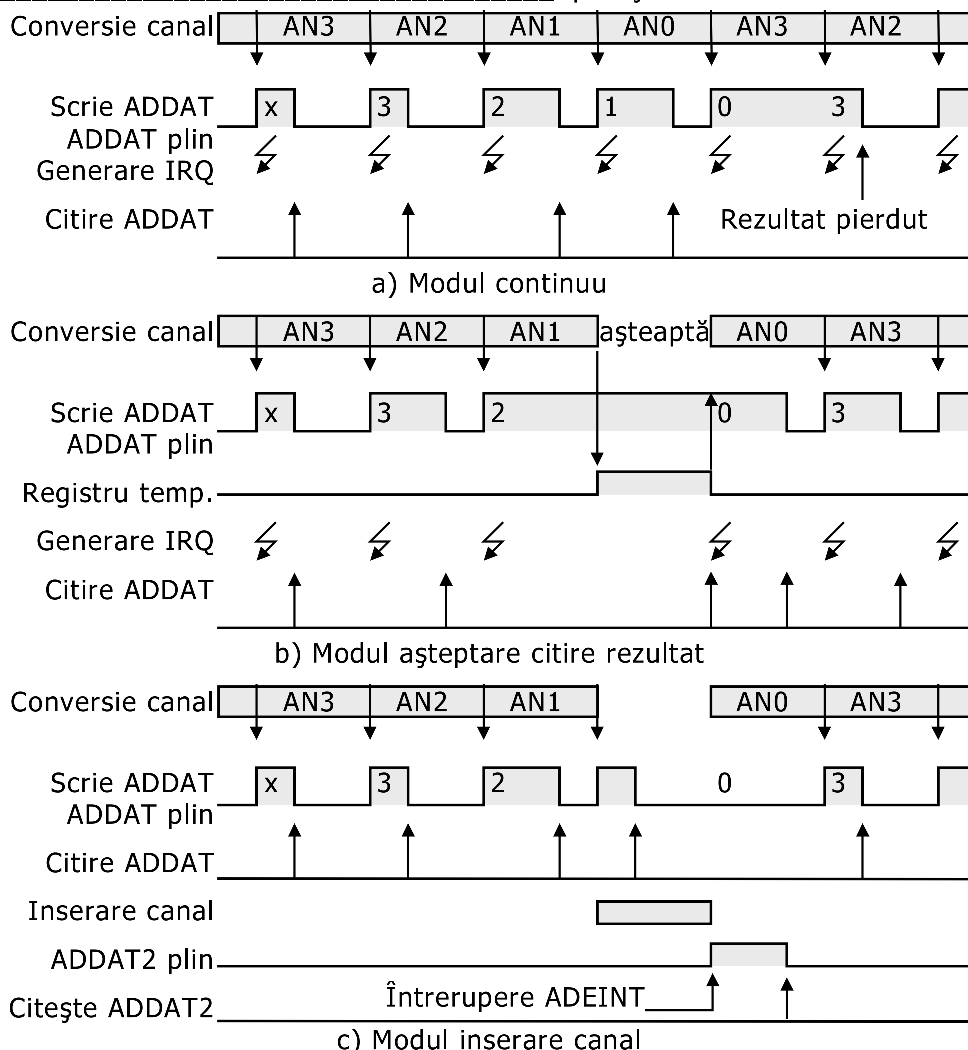


Figura 1.41. Modurile de lucru ale ADC

Întrucât condensatorul trebuie să atingă valoarea finală într-un timp cât mai scurt, trebuie ca atât rezistenţa internă a intrării analogice, cât şi a sursei de alimentare analogice să fie cât mai mici pentru a putea debita un curent cât mai mare.

Timpul necesar acestor două acţiuni (eşantionare şi conversie) poate fi programat într-un domeniu, funcţie de setarea registrului `ADCON`. Trebuie amintit totuşi, că timpul de conversie nu depinde atât de microcontroler cât de parametrii electrici ai componentelor analogice.

Semnificaţia biţilor `ADCTC` şi `ADSTC` este prezentată în tabelul 2.36.

Tabelul 1.49

ADCTC	Timpul de conversie t_{CC}	ADSTC	Timpul de eşantionare t_{SC}
00	TCL·32	00	t_{CC}
01		01	$t_{CC}\cdot 2$
10	TCL·128	10	$t_{CC}\cdot 4$
11	TCL·64	11	$t_{CC}\cdot 8$

Timpul total al conversiei este $10t_{CC} + 2t_{SC} + 4TCL$.

1.19.3 Controlul întreruperilor ADC

La sfârşitul fiecărei conversii, indicatorul `ADCIR` din registrul `ADCIC` este setat. Această cerere de întrerupere poate genera o întrerupere cu vectorul `ADCINT` sau un transfer `PEC`.

Indicatorul ADEIR din registrul ADEIC este setat numai dacă se produce o suprapunere de date în registrul ADDAT sau dacă rezultatul conversiei unui canal injectat a fost încărcat în registrul ADDAT2. Această cerere de întrerupere poate genera o întrerupere cu vectorul ADEINT sau un transfer PEC. Structura celor două registre de întrerupere este prezentată în paragraful 1.16.1 (tabelul 2.17).

1.20. Timere/Numărătoare

Timerele cu utilizare generală GPT1 şi GPT2 reprezintă o structură foarte flexibilă de numărătoare/timere care pot fi utilizate pentru sincronizări, numărare de evenimente, măsurare durate, multiplicări de frecvenţă etc.

Cele două blocuri, GPT1 şi GPT2, conţin cinci timere de 16 biţi. GPT1 conţine trei timere cu o rezoluţie maximă de 200 ns (la frecvenţa unităţii centrale de 40 MHz) în timp ce, GPT2 conţine două timere cu o rezoluţie maximă de 100 ns (la frecvenţa unităţii centrale de 40 MHz) şi un registru de 16 biţi pentru captură şi reîncărcare (CAPREL).

Fiecare timer din fiecare bloc poate lucra independent într-un număr diferit de moduri sau poate fi concatenat cu alt timer din acelaşi bloc.

1.20.1 Blocul de timere GPT1

Toate cele trei timere ale blocului (T2, T3 şi T4) pot lucra în trei moduri de bază: timer, timer comandat extern şi numărător şi fiecare timer poate număra crescător sau descrescător.

Fiecare timer are o intrare externă pentru achiziţia de semnale iar direcţia de numărare poate fi modificată atât prin program, cât şi prin intermediul unor semnale externe. De asemenea, fiecare depăşire superioară sau inferioară a timerului T3 poate fi semnalată în exterior. Timerele auxiliare T2 şi T4 pot fi concatenate cu timerul T3 sau pot fi folosite ca registre de captură sau reîncărcare pentru timerul T3. Conţinutul fiecărui timer poate fi citit sau modificat de unitatea centrală prin intermediul registrelor T2, T3 şi T4. Scrierea registrului prin program are prioritate faţă de orice altă modificare produsă de hardware.

Schema bloc a blocului de timere GPT1 este prezentată în figura 2.28.

a) Timerul T3

Timerul T3 este configurat şi controlat de registrul T3CON, descris în tabelul 2.37.

Modul timer

Acest mod este selectat pentru timerul T3 dacă câmpul T3M din registrul T3CON este egal cu 0h. Semnalul de ceas este asigurat de prescaler care divizează frecvenţa unităţii centrale cu un coeficient funcţie de câmpul T3I: $K=2^{3+T3I}$.

Figura 1.42. Timerele GPT1

Tabelul 1.50

T3CON (FF42h)	–	–	–	–	–	T3OTL	T3OE	T3UDE	T3UD	T3R	T3M			T3I		

T3OTL T3OE	Fiecare depăşire a timerului T3 comută bitul T3OTL. Dacă T3OE=1h şi P3.3 este setat ca ieşire, ieşirea T3OUT reflectă starea lui T3OTL. T3OTL poate fi utilizat pentru declanşarea încărcării timerelor T2 sau T4.			
T3UDE T3UD	Selectare direcţie numărare pentru blocul GPT1. Pinul T3EUD (P3.4) trebuie programat ca intrare.			
	Pin TxEUD	TxUDE	TxUD	Direcţie numărare
	X	0	0	Crescătoare
	X	0	1	Descrescătoare
	0	1	0	Crescătoare
	1	1	0	Descrescătoare
	0	1	1	Crescătoare
	1	1	1	Descrescătoare
T3R	Validare funcţionare timer (T3R=1h).			
T3M	'000': mod funcţionare timer; '001': mod funcţionare numărător; '010': mod funcţionare timer comandat de T3IN activ în 0 LOGIC; '011': mod funcţionare timer comandat de T3IN activ în 1 LOGIC; '1XX': Rezervat.			
T3I	Selectează modul de acţiune al intrării T3IN (mod numărător) sau coeficientul de prescalare a frecvenţei de ceas (mod timer).			

Frecvenţele de intrare în timer, rezoluţia şi perioada rezultată din coeficientul de prescalare ales sunt prezentate în tabelul 2.38. Tabelul este valabil şi pentru modurile timer comandat, precum şi pentru T2 şi T4.

Modul timer comandat

Acest mod este selectat de câmpul T3M din registrul T3CON care trebuie să aibă valoarea 2h sau 3h. Funcţionarea timerului în acest mod este similară cu cea anterioară, numai că frecvenţa de intrare este condiţionată de nivelul semnalului pe pinul T3IN (P3.6). În acest sens, pinul P3.6 trebuie setat ca intrare.

Tabelul 1.51

f_{CPU}=20 MHz	Valoare T2I, T3I, T4I							
	0h	1h	2h	3h	4h	5h	6h	7h
Coeficient divizare	8	16	32	64	128	256	512	1024
Frecvenţă intrare [kHz]	2500	1250	625	312.5	156.25	78.125	39.06	19.53
Rezoluţie [μs]	0.4	0.8	1.6	3.2	6.4	12.8	25.6	51.2
Perioadă [ms]	26	52.5	105	210	420	840	1680	3360

Bitul T3M.0 selectează nivelul activ al intrării: dacă este 0h, timerul este validat dacă T3IN are nivelul 0 LOGIC; dacă este 1h, timerul este validat dacă T3IN are nivelul 1 LOGIC.

Modul numărător

Acest mod este selectat prin setarea câmpului T3M la valoarea 1h. În acest mod sunt contorizate tranziţiile semnalelor pinului T3IN (P3.6) care trebuie setat ca intrare. Evenimentele care produc incrementări sau decrementări ale timerului pot fi fronturi crescătoare, descrescătoare sau ambele, funcţie de câmpul T3I descris în tabelul 2.39.

Tabelul 1.52

T3I	Front activ pe T3IN
000	Numărător dezactivat.
001	Front crescător.
010	Front descrescător.
011	Ambele fronturi.
1XX	Rezervat.

Frecvenţa maximă de intrare este $f_{CPU}/8$. Pentru a garanta o numărare corectă a tranziţiilor de pe pinul T3IN, nivelul semnalului trebuie să fie 0 LOGIC sau 1 LOGIC cel puţin o perioadă $8 \cdot t_{CPU}$.

b) Timerele T2 şi T4

Ambele timere au exact aceeaşi funcţionare. Ele pot funcţiona ca timere, timere comandate sau numărătoare şi au acelaşi opţiuni pentru frecvenţe şi semnalul de numărare la fel ca timerul T3.

Faţă de timerul T3, aceste două timere deţin suplimentar modul de concatenare cu timerul T3 sau pot funcţiona ca registre de captură ori reîncărcare dar nu dispun de bistabilul de ieşire T3OTL.

Structura celor două registre de control, T2CON şi T4CON este prezentată în tabelul 2.40.

Tabelul 1.53

T2CON (FF40h)	–	–	–	–	–	–	–	T2UDE	T2UD	T2R	T2M	T2I
T4CON (FF44h)	–	–	–	–	–	–	–	T4UDE	T4UD	T4R	T4M	T4I

TxUDE TxUD	Selectare direcţie numărare. Identic cu câmpurile T3UDE şi T3UD prezentate în tabelul 2.37.
TxR	Validare timer. Identic cu bitul T3R din tabelul 2.37.
TxM	'000': mod funcţionare timer; '001': mod funcţionare numărător; '010': mod funcţionare timer comandat de TxIN activ în 0 LOGIC; '011': mod funcţionare timer comandat de T3IN activ în 1 LOGIC; '100': mod funcţionare reîncărcare; '101': mod funcţionare captură; '11X': Rezervat.
TxI	Selecţie constantă prescaler (mod timer) sau fronturi active (mod numărător). În modul timer, similar cu câmpul T3I din tabelul 2.39.

Funcţionarea timerelor auxiliare T2 şi T4 în regimurile numărător sau timer este identică cu funcţionarea timerului T3.

Concatenarea timerelor T2 şi T4

Folosirea bitului T3OTL ca sursă de semnal pentru un timer auxiliar, permite concatenarea timerului T3 cu timerul T2 sau T4. Funcţie de frontul ales pentru comanda timerului auxiliar, concatenarea formează un timer sau numărător de:

- 32 de biţi, dacă ambele fronturi ale T3OTL sunt selectate să comute timerul auxiliar;
- 33 de biţi, dacă numai front crescător sau descrescător al T3OTL este selectat să comute timerul auxiliar.

Direcţiile de numărare ale celor două timere pot fi diferite, permiţând o mare varietate de configuraţii.

Reîncărcarea timerului T3

Acest regim este selectat prin setarea câmpului TxM din registrul TxCON cu valoarea 4h. În acest mod, timerul T3 este încărcat cu conţinutul unui timer auxiliar, condiţionat fie de o comutare a intrării T3OTL, fie a intrării timerului auxiliar TxIN.

Atenţie! Dacă este utilizat ca registru de reîncărcare, timerul auxiliar se opreşte automat, indiferent de valoarea bitului TxR.

Dacă este folosit pentru declanşarea încărcării tranziţia semnalului T3OTL se va declanşa o întrerupere T3IR.

Trebuie evitată folosirea aceluiaşi eveniment pentru ambele timere auxiliare, unitatea centrală încercând să încarce valorile din ambele registre. În acest caz valoarea T2 este neglijată şi este încărcată valoarea din T3.

Sub controlul T3OTL sunt posibile mai multe configuraţii de reîncărcare, funcţie de fronturile active utilizate:

- Dacă sunt selectate ambele fronturi ale tranziţiei T3OTL, timerul T3 va fi reîncărcat la fiecare depăşire superioară sau inferioară cu valoarea din timerul auxiliar. Este modul implicit de funcţionare în acest regim.
- Dacă este selectat un singur front al T3OTL, timerul T3 va fi reîncărcat la fiecare a doua depăşire.
- Folosind modul cu un singur front al T3OTL pentru ambele timere auxiliare (o încărcare dintr-un timer auxiliar declanşată de frontul crescător, în timp ce cealaltă încărcare, din celălalt timer auxiliar, declanşată de frontul descrescător) este posibilă realizarea unui modulator de impulsuri în durată extrem de flexibil.

Capturarea valorii timerului T3

Acest regim este selectat prin setarea câmpului TxM din registrul TxCON cu valoarea 5h.

Acest mod presupune încărcarea valorii curente a timerului T3 într-un timer auxiliar ca răspuns la tranziţia semnalului pe pinul extern TxIN. Semnalul de declanşare poate fi un front crescător, descrescător sau ambele. Selectarea tranziţiei este făcută de biţii mai puţin semnificativi din registrul TxI (valoarea exactă este indicată în tabelele 2.39 şi 2.40).

Atenţie! Dacă este utilizat ca registru de captură, timerul auxiliar se opreşte automat, indiferent de valoarea bitului TxR.

Biţii de control ai direcţiei pentru T2IN şi T4IN (DP3.7, respectiv DP3.5) trebuie şterşi.

Nivelul semnalului de pe pinii T2IN sau T4IN trebuie să-şi păstreze starea cel puţin $8t_{CPU}$.

1.20.2 Blocul de timere GPT2

Ambele timere ale blocului (T5 şi T6) pot lucra în trei moduri de bază: timer, timer comandat extern şi numărător şi fiecare timer poate număra crescător sau descrescător. Rezoluţia maximă a acestor timere este de 100 ns (pentru o frecvenţă a procesorului de 40 MHz).

Fiecare timer are o intrare externă pentru achiziţia de semnale iar direcţia de numărare poate fi modificată atât prin program, cât şi prin intermediul unor semnale externe. De asemenea, fiecare depăşire superioară sau inferioară a timerului T6 poate fi semnalată în exterior.

Timerul auxiliar T6 poate fi concatenat cu timerul T5 dar T5 poate fi concatenat şi cu timerele modulului CAPCOM prin intermediul unei conexiuni.

Valoarea registrului timerului T5 poate fi capturată în registrul de 16 biţi CAPREL şi, opţional, poate fi ştearsă; timerul T6 poate fi reîncărcat prin intermediul aceluiaşi registru CAPREL.

Conţinutul fiecărui timer poate fi citit sau modificat de unitatea centrală prin intermediul registrelor T5 şi T6. Scrierea registrului prin program are prioritate faţă de orice altă modificare produsă de hardware.

Schema bloc a blocului de timere GPT2 este prezentată în figura 2.29.

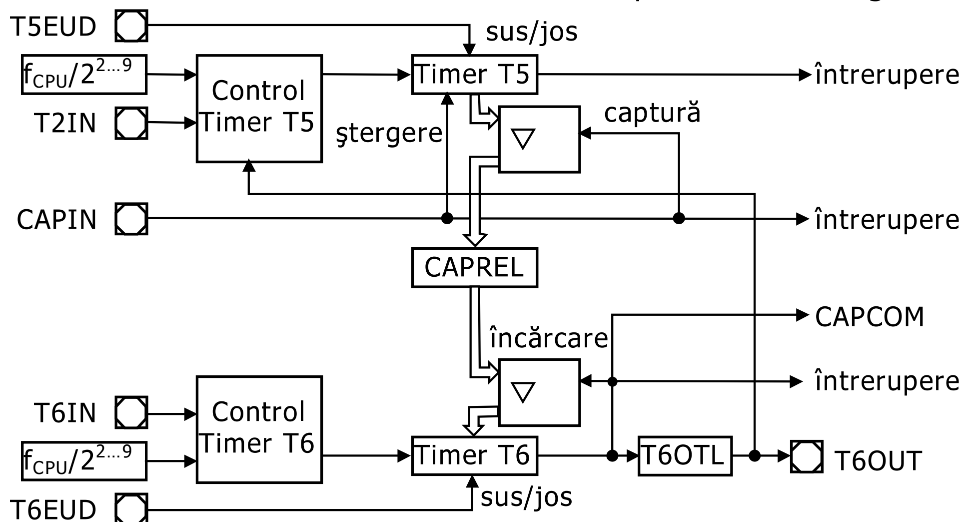


Figura 1.43. Timerele GPT2

a) Timerul T6

Timerul T6 este configurat şi controlat de registrul T6CON, descris în tabelul 2.41.

<table>
<tr><td colspan="17" align="right">Tabelul 1.54</td></tr>
<tr><td>T6CON
(FF42h)</td><td>T6SR</td><td>–</td><td>–</td><td>–</td><td>–</td><td>T6OTL</td><td>T6OE</td><td>T6UDE</td><td>T6UD</td><td>T6R</td><td colspan="3">T6M</td><td colspan="3">T6I</td></tr>
<tr><td>T6SR</td><td colspan="16">Validare regim reîncărcare din registrul CAPREL.</td></tr>
<tr><td>T6OTL
T6OE</td><td colspan="16">Fiecare depăşire a timerului T6 comută bitul T6OTL. Dacă T6OE=1h şi P3.1 este setat ca ieşire, ieşirea T6OUT reflectă starea lui T6OTL. Dacă T6OE este şters, P3.1 poate fi utilizat ca pin de port de intrare-ieşire.
T6OTL poate fi utilizat ca intrare pentru T5 în regim numărător.</td></tr>
<tr><td rowspan="8">T6UDE
T6UD</td><td colspan="16">Selectare direcţie numărare pentru blocul GPT2. Pinul T6EUD (P5.10) trebuie programat ca intrare.</td></tr>
<tr><td colspan="4">Pin TxEUD</td><td colspan="4">TxUDE</td><td colspan="4">TxUD</td><td colspan="4">Direcţie numărare</td></tr>
<tr><td colspan="4">X</td><td colspan="4">0</td><td colspan="4">0</td><td colspan="4">Crescătoare</td></tr>
<tr><td colspan="4">X</td><td colspan="4">0</td><td colspan="4">1</td><td colspan="4">Descrescătoare</td></tr>
<tr><td colspan="4">0</td><td colspan="4">1</td><td colspan="4">0</td><td colspan="4">Crescătoare</td></tr>
<tr><td colspan="4">1</td><td colspan="4">1</td><td colspan="4">0</td><td colspan="4">Descrescătoare</td></tr>
<tr><td colspan="4">0</td><td colspan="4">1</td><td colspan="4">1</td><td colspan="4">Crescătoare</td></tr>
<tr><td colspan="4">1</td><td colspan="4">1</td><td colspan="4">1</td><td colspan="4">Descrescătoare</td></tr>
<tr><td>T6R</td><td colspan="16">Validare funcţionare timer (T6R=1h).</td></tr>
<tr><td>T6M</td><td colspan="16">‘000’: mod funcţionare timer;
‘001’: mod funcţionare numărător;
‘010’: mod funcţionare timer comandat de T6IN activ în 0 LOGIC;
‘011’: mod funcţionare timer comandat de T6IN activ în 1 LOGIC;
‘1XX’: Rezervat.</td></tr>
<tr><td>T6I</td><td colspan="16">Selectează frontul activ al intrării T6IN (mod numărător) sau coeficientul de prescalare a frecvenţei de ceas (mod timer).</td></tr>
</table>

Modul timer

Acest mod este selectat pentru timerul T6 dacă câmpul T6M din registrul T6CON este egal cu 0h. Semnalul de ceas este asigurat de prescaler care divizează frecvenţa unităţii centrale cu un coeficient funcţie de câmpul T6I: $K=2^{2+T6I}$.

Frecvenţele de intrare în timer, rezoluţia şi perioada rezultată din coeficientul de prescalare ales sunt prezentate în tabelul 2.42. Tabelul este valabil şi pentru modurile timer comandat, precum şi pentru T5.

Tabelul 1.55

f_{CPU}=20 MHz	Valoare T5I, T6I							
	0h	1h	2h	3h	4h	5h	6h	7h
Prescalare	8	16	32	64	128	256	512	1024
Frecvenţă intrare [kHz]	5000	2500	1250	625	312.5	156.25	78.125	39.06
Rezoluţie [μs]	0.2	0.4	0.8	1.6	3.2	6.4	12.8	25.6
Perioadă [ms]	13	26	52.5	105	210	420	840	1680

Modul timer comandat

Acest mod este selectat de câmpul T6M din registrul T6CON care trebuie să aibă valoarea 2h sau 3h. Funcţionarea timerului în acest mod este similară cu cea anterioară, numai că frecvenţa de intrare este condiţionată de nivelul semnalului pe pinul T6IN (P5.12). În acest sens, pinul P5.12 trebuie setat ca intrare.

Bitul T6M.0 selectează nivelul activ al intrării: dacă este 0h, timerul este validat dacă T6IN are nivelul 0 LOGIC; dacă este 1h, timerul este validat dacă T6IN are nivelul 1 LOGIC.

Modul numărător

Acest mod este selectat prin setarea câmpului T6M la valoarea 1h. În acest mod sunt contorizate tranziţiile semnalelor pinului T6IN (P5.12) care trebuie setat ca intrare. Evenimentele care produc incrementări sau decrementări ale numărătorului pot fi fronturi crescătoare, descrescătoare sau ambele, funcţie de câmpul T6I descris în tabelul 2.43.

Tabelul 1.56

T6I	Front activ pe T6IN
000	Numărător dezactivat.
001	Front crescător.
010	Front descrescător.
011	Ambele fronturi.
1XX	Rezervat.

Frecvenţa maximă de intrare este $f_{CPU}/4$. Pentru a garanta o numărare corectă a tranziţiilor de pe pinul T6IN, nivelul semnalului trebuie să fie 0 LOGIC sau 1 LOGIC cel puţin o perioadă $4 \cdot t_{CPU}$.

b) Timerul T5

Timerul T5 poate funcţiona ca timer, timer comandat sau numărător şi are acelaşi opţiuni pentru frecvenţă şi semnalul de numărare la fel ca timerul T6.

Structura registrului de control T5CON este prezentată în tabelul 2.44.

Funcţionarea timerului auxiliar T5 în regimurile numărător sau timer este identică cu funcţionarea timerului T6.

Concatenarea timerelor T5 şi T6

Folosirea bitului T6OTL ca sursă de semnal pentru timerul auxiliar, permite concatenarea timerului T6 cu timerul T5. Funcţie de frontul ales pentru comanda timerului auxiliar, concatenarea formează un timer sau numărător de:

- 32 de biţi, dacă ambele fronturi ale T6OTL sunt selectate să comute timerul auxiliar;
- 33 de biţi, dacă numai front crescător sau descrescător al T6OTL este selectat să comute timerul auxiliar.

Direcţiile de numărare ale celor două timere pot fi diferite, permiţând o mare varietate de configuraţii.

Reîncărcarea timerului T6

Acest regim este selectat prin setarea bitului T6SR din registrul T6CON. În acest mod, timerul T6 este încărcat cu conţinutul registrului CAPREL, condiţionat de o depăşire la numărare a timerului T6, simultan cu declanşarea unei întreruperi T6IR.

Tabelul 1.57

T5CON (FF46h)	T5SC	T5CLR	CI		–	–	–	T5UDE	T5UD	T5R	–	T5M		T5I		

T5SC	Validare captură în registrul CAPREL.
T5CLR	Activare ştergere timer la capturare.
CI	'00': captura dezactivată; '01': captură pe front crescător a semnalului CAPIN; '10': captură pe front descrescător a semnalului CAPIN; '11': captură pe ambele fronturi ale semnalului CAPIN;
T5UDE T5UD	Selectare direcţie numărare. Identic cu câmpurile T6UDE şi T6UD prezentate în tabelul 2.41.
T5R	Validare timer.
T5M	'00': mod funcţionare timer; '01': mod funcţionare numărător; '10': mod funcţionare timer comandat de T5IN activ în 0 LOGIC; '11': mod funcţionare timer comandat de T5IN activ în 1 LOGIC.
T5I	Selecţie constantă prescaler (mod timer) sau fronturi active (mod numărător). În modul timer, similar cu câmpul T6I din tabelul 2.42. Pentru modul numărător are următoarea semnificaţie. 'x00': numărătorul T5 dezactivat; '001': front crescător pe T5IN; '010': front descrescător pe T5IN; '011': orice tranziţie pe T5IN; '101': front crescător pe T6OTL; '110': front descrescător pe T6OTL; '111': orice tranziţie pe T6OTL.

Capturarea valorii timerului T5

Acest regim este selectat prin setarea bitului T5SC din registrul T5CON.

Acest mod presupune încărcarea valorii curente a timerului T5 în registrul CAPREL ca răspuns la tranziţia semnalului pe pinul extern CAPIN, simultan cu setarea indicatorului de întrerupere CRIR. Acelaşi eveniment poate şterge conţinutul timerului T5 după încărcarea sa, dacă bitul T5CLR din registrul T5CON este setat.

Semnalul de declanşare poate fi un front crescător, descrescător sau ambele. Selectarea tranziţiei este făcută de câmpul CI din registrul T5CON.

Atenţie! Nivelul semnalului de pe pinul CAPIN trebuie să-şi păstreze starea cel puţin $4t_{CPU}$.

Multiplicarea frecvenţei

Deoarece funcţiile de reîncărcare şi captură a registrului CAPREL pot fi validate individual de biţii T5SC şi T6SR, cele două funcţii pot fi setate simultan. Această facilitate a GPT2 permite generarea unei frecvenţe de ieşire care este un multiplu al frecvenţei de intrare.

Acest regim poate fi implementat în modul următor:

- timerul T5 funcţionează în mod numărător, crescător, la o frecvenţă suficientă pentru rezoluţia dorită, fie aceasta $f_{CPU}/32$;
- semnalul de multiplicat este aplicat pe pinul CAPIN;
- când survine o tranziţie a semnalului CAPIN, valoarea T5 este încărcată în CAPREL şi T5 este şters; în acest mod, CAPREL va conţine valoarea timpului între două evenimente externe, măsurat în incremente de T5;
- timerul T6 funcţionează în mod numărător, descrescător, la o frecvenţă funcţie de multiplicarea dorită, fie aceasta $f_{CPU}/4$; T6 este încărcat cu valoarea CAPREL la depăşirea inferioară. Aceasta înseamnă că valoarea din CAPREL reprezintă timpul între două depăşiri a T6 măsurat în incremente de 8 ori mai rapide decât ale timerului T5;
- fiecare depăşire a T6 setează indicatorul de întrerupere T6IR şi poate fi obţinut în exterior pe pinul T6OUT, având o frecvenţă de 8 ori mai mare decât a semnalului de pe CAPIN.

În acest mod, relativ simplu, fără a fi necesară nici o componentă externă, este realizat multiplicatorul de frecvenţă.

1.20.3 Întreruperile blocurilor de timere GPT1 şi GPT2

Ori de câte ori survine o depăşire FFFFh→0000h sau 0000h→FFFFh la oricare timer sau numărător, se setează un indicator de întrerupere T2IR…T6IR din registrul TxIC corespunzător. Aceasta va produce o întrerupere cu vectorii TxINT sau o cerere de serviciu PEC dacă indicatorii de validare a întreruperii TxIE sunt setaţi.

Suplimentar, la blocul GPR2 poate fi generată o întrerupere la tranziţia semnalului CAPIN cu sensul corespunzător cu setarea câmpului CI din registrul T5CON. Aceasta produce setarea indicatorului CRIR din registrul

CRIC şi generarea unei întreruperi cu vectorul CRINT sau o cerere de serviciu PEC.

Structura registrelor de control a întreruperilor TxIC şi CRIC este prezentată în paragraful 1.16.1, tabelul 2.17.

1.21. Comparatoarele şi registrele de captură

Circuitul 80C167 dispune de două module de captură şi comparare (CAPCOM) aproape identice care constau în 32 de registre care interacţionează cu 4 timere.

Modulele CAPCOM pot *captura* conţinutul unui timer condiţionat de un eveniment intern sau extern sau pot *compara* conţinutul unui timer cu o valoare dată şi, în caz de egalitate, să modifice starea unor semnale externe. Acest mecanism permite generarea şi controlul unor secvenţe temporale de până la 16 canale pentru fiecare modul, fără a necesita circuite externe şi cu un minim de intervenţie a programului.

Astfel, modulele CAPCOM pot fi utilizate pentru operarea unor evenimente externe extrem de rapide cum ar fi: generarea de forme de undă şi impulsuri, modularea în durată a impulsurilor, înregistrarea momentului la care survin diferite evenimente externe etc. De asemenea, modulele CAPCOM permit implementarea a cel mult 16 timere software.

Rezoluţia maximă a modulelor CAPCOM a unui circuit cu frecvenţa de ceas de 40 MHz este de 200 ns.

Fiecare modul CAPCOM conţine două timere de 16 biţi (T0 şi T1 pentru CAPCOM1, respectiv T7 şi T8 pentru CAPCOM2), două registre de reîncărcare (TxREL), şi un banc de 16 registre de captură/reîncărcare (CC0…CC15 în CAPCOM1, respectiv CC16…CC31 în CAPCOM2).

Frecvenţa de intrare în modulele CAPCOM este programabilă, fie provenită din frecvenţa unităţii centrale (divizată cu un coeficient programabil), fie derivată din depăşirile timerului T6 din blocul GPT2. T0 şi T7 pot opera şi în modul numărător, permiţând contorizarea unor evenimente externe.

Fiecare registru de comparare/reîncărcare poate fi programat individual pentru una din funcţiuni şi poate fi asociat unuia din cele două timere ale modulului. Fiecare registru de comparare/reîncărcare dispune de un pin de intrare-ieşire asociat (cu excepţia CC24…CC27 pe pinii P1H.4…P1H.7 care nu au decât funcţii de captură).

Sunt generate întreruperi specifice pentru fiecare eveniment de capturare/comparare sau depăşiri ale timerelor.

Schema bloc a modulelor CAPCOM este prezentată în figura 2.30.

1.21.1 Timerele CAPCOM

Utilizarea primordială a timerelor T0/T1 şi T7/T8 este de a asigura două baze de timp independente pentru registrele de captură/comparare asociate dar este posibilă şi folosirea timerelor independent de registrele respective.

Funcţionarea timerelor CAPCOM este controlată de registrele T01CON şi T78CON, ambele registre având opţiuni identice pentru toate cele 4 timere.

Structura celor două registre este prezentată în tabelul 2.45.

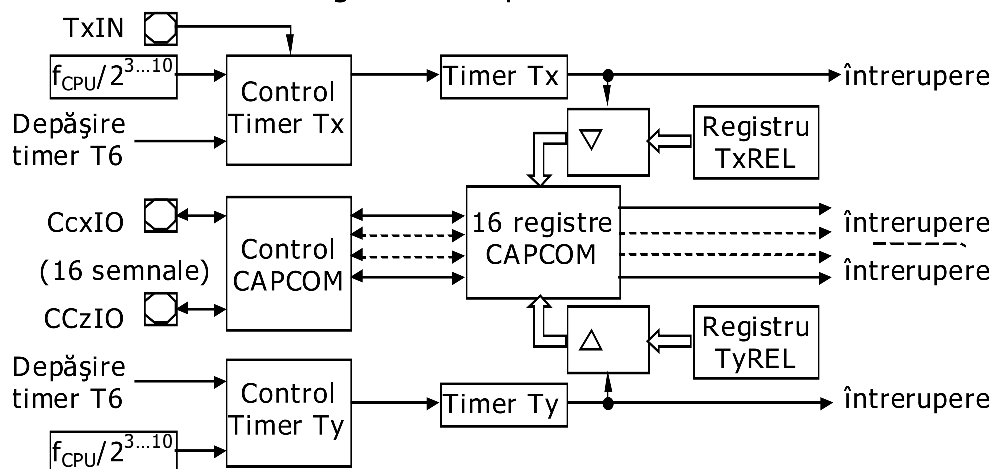


Figura 1.44. Schema bloc a modulelor CAPCOM

Tabelul 1.58

T01CON (FF50h)	–	T1R	–	–	T1M	T1I	–	T0R	–	–	T0M	T0I
T78CON (FF20h)	–	T8R	–	–	T8M	T8I	–	T7R	–	–	T7M	T7I
TxR	Validare funcţionare timer Tx.											
TxM	0h: mod timer (semnal de intrare derivat din ceasul sistem); 1h: mod numărător (semnale de intrare extern sau depăşiri timer T6).											
TxI	Coeficient divizare f_{CPU} (pentru TxM=0): $K=2^{TxI+3}$. Pentru TxM=1: 'X00': numărare depăşiri timer T6; 'X01': numărare fronturi crescătoare ale T0IN sau T7IN; 'X10': numărare fronturi descrescătoare ale T0IN sau T7IN; 'X11': numărare ambele fronturi ale T0IN sau T7IN.											

În toate modurile, timerele funcţionează numai în sens crescător. Unitatea centrală poate citi sau modifica orice registru Tx şi la scriere are prioritate asupra modificărilor produse de hardware.

Modul timer

Acest mod este selectat pentru timerele Tx dacă biţii TxM din registrele T01CON şi T78CON sunt şterşi. Semnalul de ceas este asigurat de prescaler care divizează frecvenţa unităţii centrale cu un coeficient funcţie de câmpul TxI: $K=2^{3+TxI}$.

Frecvenţele de intrare în timer, rezoluţia şi perioada rezultată din coeficientul de prescalare ales sunt prezentate în tabelul 2.38.a).

Modul numărător

Acest mod este selectat prin setarea câmpului TxM. În acest mod sunt contorizate depăşirile produse de timerul T6 sau de tranziţiile semnalelor

pinului T0IN (P3.0), respectiv T7IN (P2.15) care trebuie setate ca intrări. Dacă timerele T1 şi T8 sunt folosite ca numărătoare, este obligatorie setarea câmpului TxI cu valoarea X00h; în caz contrar timerele T1 sau T8 sunt dezactivate.

Frecvenţa maximă de intrare este $f_{CPU}/16$. Pentru a garanta o numărare corectă, nivelul semnalului trebuie să fie 0 LOGIC sau 1 LOGIC cel puţin o perioadă $8 \cdot t_{CPU}$.

Modul reîncărcare

Încărcarea registrelor Tx se execută de fiecare dată când se produce o depăşire, atât în modul timer, cât şi în modul numărător. Valoarea de încărcare este păstrată în registrul TxREL.

1.21.2 Registrele captură şi comparare

Cele 32 de registre de captură/comparare CC0…31 sunt folosite ca registre de date pentru operaţiile de captură sau comparare pentru timerele T0/T1, respectiv T7/T8.

Fiecare registru poate fi programat individual în modul captură sau în patru moduri diferite de comparare (cu excepţia CC24…27), putând fi alocate unuia sau ambelor timere dintr-un modul.

Orice registru de captură/comparare poate fi folosit ca registru cu utilizare generală în situaţia în care el nu este utilizat pentru funcţiile sale specifice modulului CAPCOM.

Controlul celor 32 de registre de captură/comparare este asigurat de 8 registre CCM0…CCM7, organizate identic. Structura lor este prezentată în tabelul 2.46.

Tabelul 1.59

CCM0 (FF52h)	ACC3	CCMOD3	ACC2	CCMOD2	ACC1	CCMOD1	ACC0	CCMOD0
CCM1 (FF54h)	ACC7	CCMOD7	ACC6	CCMOD6	ACC5	CCMOD5	ACC4	CCMOD4
CCM2 (FF56h)	ACC11	CCMOD11	ACC10	CCMOD10	ACC9	CCMOD9	ACC8	CCMOD8
CCM3 (FF58h)	ACC15	CCMOD15	ACC14	CCMOD14	ACC13	CCMOD13	ACC12	CCMOD12
CCM4 (FF22h)	ACC19	CCMOD19	ACC18	CCMOD18	ACC17	CCMOD17	ACC16	CCMOD16
CCM5 (FF24h)	ACC23	CCMOD23	ACC22	CCMOD22	ACC21	CCMOD21	ACC20	CCMOD20
CCM6 (FF26h)	ACC27	CCMOD27	ACC26	CCMOD26	ACC25	CCMOD25	ACC24	CCMOD24
CCM7 (FF28h)	ACC31	CCMOD31	ACC30	CCMOD30	ACC29	CCMOD29	ACC28	CCMOD28

ACCx	0h: CCx alocat T0/T7; 1h: CCx alocat T1/T8	
CCMODx	000: Modul CC dezactivat; 001: Captură front ↑ pin CCxIO; 010: Captură front ↓ pin CCxIO; 011: Captură fronturi ↑ ↓ pin CCxIO.	100: Comparare mod 0; 101: Comparare mod 1; 110: Comparare mod 2; 111: Comparare mod 3.

Atenţie! Un eveniment de captură/comparare al canalului CC31 poate fi folosit pentru declanşarea inserării unui canal al modulului ADC (prezentat în paragraful 1.19.d).

a) Modul captură

Ca răspuns la un eveniment extern, conţinutul timerelor T0/T1 sau T7/T8 este încărcat în registrul de captură alocat, funcţie de modulul CAPCOM utilizat şi de valoarea bitului ACCx.

Evenimentul extern care produce capturarea registrului Tx este un front crescător, descrescător sau ambele ale intrării CCxIO şi poate fi selectat prin program. Evenimentul care produce captura setează indicatorul corespunzător CCxIR, generând o întrerupere cu vectorul CCxINT sau o cerere de transfer PEC.

Atenţie! Pinul folosit pentru evenimentul extern CCxIO trebuie setat ca intrare. Dacă este setat ca ieşire, funcţia de captură se declanşează numai prin program, modificând valoarea bitului, în scopuri de testare.

Pentru a garanta interceptarea corectă a semnalului extern, este obligatoriu ca acesta să-şi păstreze starea cel puţin $8 \cdot t_{CPU}$.

b) Modurile de comparare

Modurile de comparare permit declanşarea unor evenimente externe sincronizate cu valoarea registrelor numărătoarelor CAPCOM.

Valoarea memorată în registrele CCx este comparată în permanenţă cu conţinutul timerului alocat (T0/T1 sau T7/T8) şi, în cazul egalităţii, este generat un semnal extern pe pinul CCxIO sau este setat indicatorul de întrerupere CCxIR.

Când oricare două registre de comparare sunt setate la aceiaşi valoare, indicatorii lor de întreruperi sunt setaţi simultan, iar semnalul de ieşire programat va fi generat la $8 \cdot t_{CPU}$ după ce timerul urmărit a atins valoarea de comparare. Mai departe, chiar dacă timerul este incrementat sau setat prin program, compararea cu valoarea respectivă este dezactivată.

Cele patru moduri de comparare posibile la circuitul 80C167 sunt selectate prin intermediul câmpului CCMODx din registrul CCMx.

Modul de comparare 0

Acest regim generează numai o întrerupere, utilizabilă în scopuri de sincronizare a programului. Modul 0 pentru registrul CCx este selectat prin setarea câmpului CCMODx din registrul CCMx la valoarea 4h.

În acest mod indicatorul de întrerupere CCxIR este setat de fiecare dată când registrele CCx şi Tx au aceiaşi valoare. Modul de comparare 0 permite generarea a mai multor cereri de întrerupere datorită posibilităţii modificării conţinutului registrului CCx pentru acelaşi ciclu de numărare al timerului (perioada până la care timerul generează o depăşire).

Pinul corespunzător CCxIO nu este afectat şi poate fi utilizat ca pin de intrare-ieşire.

Modul de comparare 1

Modul 1 este selectat prin setarea câmpului CCMODx la valoarea 5h.

Când valoarea registrului timerului devine egală cu valoarea registrului de comparare, acest regim de funcţionare setează indicatorul de întrerupere CCxIR şi comută ieşirea CCxIO. Comutarea ieşirii se face prin citirea bistabilului de ieşire, inversarea sa şi rescrierea lui în bistabil.

De asemenea, acest mod permite modificarea conţinutului registrului CCx pe parcursul unui ciclu de timer.

Atenţie! Dacă bistabilul de ieşire al portului este scris simultan cu evenimentul de comparare, va avea prioritate programul.

Canalele CC24…27 vor genera numai întreruperile corespunzătoare fără a modifica pinii de ieşire.

Valoarea iniţială a ieşirii comutate poate fi aleasă prin program.

Modul de comparare 2

Modul 2 este selectat prin setarea câmpului CCMODx la valoarea 6h.

Funcţionează asemănător cu modul 0 generând numai întreruperi, cu excepţia faptului că în acest regim nu se pot genera mai multe întreruperi pentru acelaşi ciclu de timer.

Modul de comparare 3

Modul 3 este selectat prin setarea câmpului CCMODx la valoarea 7h.

Funcţionează asemănător cu modul 1 generând întreruperi şi comutând ieşirea CCxIO, numai că în acest regim nu se pot genera mai multe întreruperi pentru acelaşi ciclu de timer. De asemenea, atât indicatorul de întrerupere CCxIR cât şi ieşirea CCxIO sunt setate de concordanţa între registre şi sunt şterse când timerul alocat generează o depăşire.

În situaţia în care registrul de reîncărcare TxREL este identic cu registrul de comparare CCx, se generează numai întreruperea iar semnalul de ieşire este nemodificat.

Modul de comparare cu registru dublu

În acest mod două registre CCx lucrează împreună pentru a controla un singur pin de ieşire.

Astfel, registrele CC0…CC7 şi CC16…CC23 formează bancul 1 de registre, în timp ce registrele CC8…CC15 şi CC24…CC23 formează bancul 2 de registre. În acest mod, un registru din bancul 1 şi un registru din bancul 2 formează un registru pereche şi controlează pinul de ieşire alocat bancului 1.

Acest regim de lucru poate fi selectat individual pentru fiecare registru pereche: registrul din bancul 1 setat în modul de comparare 1 iar registrul din bancul 2 setat în modul 0.

La detectarea egalităţii între Tx şi unul din registrele pereche, este setat indicatorul de întrerupere corespunzător registrului CCx iar pinul CCxIO este comutat. Dacă registrele pereche au aceiaşi valoare, sunt generate simultan două cereri de întrerupere, corespunzătoare celor două registre CCx, în timp ce ieşirea CCxIO este comutată o singură dată.

1.21.3 Întreruperile modulului CAPCOM

Modulul CAPCOM generează două categorii de întreruperi:

- 4 întreruperi generate de depăşirile timerelor T0, T1, T7 şi T8;
- 32 de întreruperi generate de evenimentele de captură/comparare ale registrelor CC0…CC31.

În mod corespunzător, sunt generate întreruperi cu vectorii T0INT, T1INT, T7INT, T8INT sau CC0INT…CC31INT sau sunt solicitate transferuri PEC.

Structura registrelor de control a întreruperilor TxIC şi CCxIC sunt prezentate în tabelul 2.17 din paragraful 1.16.1.

1.22. Timerul de iniţializare – watchdog

Pentru a permite restabilirea sistemului după o eroare hardware sau software, circuitul 80C167 are prevăzut un timer de iniţializare, denumit în continuare watchdog timer (WDT).

Rolul acestuia este de a reseta unitatea centrală dacă datorită unei blocări a sistemului, programul de aplicaţie nu serveşte unitatea WDT. Această procedură previne o funcţionare defectuoasă pentru o durată mai mare decât cea prestabilită pentru reiniţializare.

Structura internă a blocului watchdog este prezentată în figura 2.31.

Figura 1.45. Blocul watchdog

Modulul WDT constă într-un timer de 16 biţi care este comandat fie cu $f_{CPU}/2$, fie cu $f_{CPU}/128$. Timerul este realizat prin concatenarea a două timere de 8 biţi, WDTL şi WDTH. Timerul WDTH poate fi setat de utilizator pentru a selecta durata de acţiune a timerului.

Funcţionarea WDT este controlată de registrul WDTCON, prezentat în tabelul 2.47.

Tabelul 1.60

WDTCON (FFAEh)	WDTREL								–	–	–	–	–	–	WDTR	WDTIN

WDTREL	Valoarea de reîncărcare a octetului superior.
WDTR	Setat de WDT la depăşire.
WDTIN	0h: frecvenţă de intrare $f_{CPU}/2$; 1h: frecvenţă de intrare $f_{CPU}/128$.

După orice iniţializare software (instrucţiunea SRST), hardware sau WDT, modulul watchdog este validat şi începe incrementarea de la 0000h cu frecvenţa $f_{CPU}/2$. Prin program, prin intermediul bitului WDTIN, frecvenţa de incrementare se poate modifica la $f_{CPU}/128$.

Timerul WDT poate fi oprit prin instrucţiunea DISWDT care nu este executată decât în intervalul dintre iniţializare şi instrucţiunile EINIT (sfârşit iniţializare) sau SRVWDT (servire WDT).

Dacă WDT nu este dezactivat, el continuă să numere, chiar în modul inactiv (prezentat în paragraful 1.26). Dacă nu este reîncărcat prin instrucţiunea SRVWDT până în momentul în care a ajuns la valoarea FFFFh, la următoarea incrementare WDT va declanşa iniţializarea sistemului.

Atenţie! După o iniţializare hardware care activează încărcătorul bootstrap (prezentat în paragraful 1.25.), WDT va fi dezactivat.

Perioada de acţiune a blocului WDT poate fi programată în două moduri: fie prin selectarea frecvenţei de numărare (WDTIN), fie prin modificarea valorii de reîncărcare (WDTREL).

Perioada între servirea WDT şi declanşarea unei iniţializări se poate determina cu relaţia:

$$T_{WDT} = \frac{2^{(1+WDTIN \cdot 6)} \cdot \left(2^{16} - WDTREL \cdot 2^{8}\right)}{f_{CPU}}$$

Orientativ, sunt prezentate valorile maxime şi minime ale T_{WDT} (în milisecunde) pentru un procesor cu f_{CPU}=20 MHz.

WDTREL	WDTIN=0	WDTIN=1
FFh	25.6	1.6
00h	6.55	419

1.23. Interfaţa serială asincronă/sincronă

Interfaţa serială asincronă/sincronă, denumită în continuare ASC0 permite comunicarea serială între circuitul 80C167 şi alte microcontrolere, microprocesoare sau alte periferice externe.

ASC0 admite o comunicaţie full-duplex asincronă până la o viteză de 625 KBaud sau half-duplex sincronă până la o viteză de 2.5 MBaud (pentru unităţi centrale cu frecvenţa de ceas de 20 MHz).

În mod asincron, datele sunt transferate în formatul de 8 sau 9 biţi, cu generarea bitului de paritate şi un număr de biţi de stop selectabili. Pentru creşterea siguranţei transmisiei, sunt detectate erorile de paritate, depăşire şi încadrare. Emisia şi recepţia sunt dublu bufferate. Pentru generarea ratei de transmisie, ASC0 dispune de un timer de 13 biţi. Pentru scopuri de testare, ASC0 are un regim de funcţionare în buclă închisă.

În mod sincron, datele sunt transmise sincronizate faţă de un ceas de deplasare asigurat de modul. Pentru modul multiprocesor, ASC0 dispune de un mecanism pentru separarea datelor de adrese.

Controlul modulului ASC0 este asigurat de registrul S0CON prezentat în tabelul 2.48.

Tabelul 1.61

S0CON (FFB0h)	S0R	S0LB	S0BRS	S0ODD	–	S0OE	S0FE	S0PE	S0OEN	S0FEN	S0PEN	S0REN	S0STP		S0M	

S0R	0h: generator rată de transmisie oprit (ASC0 dezactivat); 1h: generator rată transmisie validat.
S0LB	0h: mod de lucru standard; 1h: mod de lucru în buclă.
S0BRS	0h: generare rată transmisie în mod standard; 1h: reducere rată de transmisie cu 2/3.
S0ODD	0h: validare paritate pară; 1h: validare paritate impară.
S0OE S0FE S0PE	Indicatori întrerupere eroare depăşire, încadrare, respectiv paritate.
S0OEN S0FEN S0PEN	0h: ignorare erori depăşire, încadrare, respectiv paritate; 1h: verificare erori depăşire, încadrare, respectiv paritate.
S0REN	0h: recepţie dezactivată; 1h: recepţie validată.
S0STP	0h: un bit de stop; 1h: doi biţi de stop.
S0M	'000': date 8 biţi (mod sincron); '001': date 8 biţi (mod asincron); '010': rezervat; '011': date 7 biţi + paritate (mod asincron); '100': date 9 biţi (mod asincron); '101': date 8 biţi + bit atenţionare (mod asincron); '110': rezervat; '111': date 8 biţi + paritate (mod asincron).

O transmisie este iniţiată prin scrierea registrului de emisie S0TBUF prin intermediul unei instrucţiuni sau a unui transfer PEC. După ce o transmisie a fost finalizată, registrul S0TBUF este şters.

Transmisia datelor este dublu bufferată, adică un nou caracter poate fi scris în registru înainte ca transmisia primului caracter să fie completă. Acest mod de lucru permite transmiterea caracterelor unul după altul, fără intervale.

Recepţia datelor este validată de bitul S0REN. După ce recepţia unui caracter a fost încheiată, datele şi, eventual, bitul de paritate pot fi citite din registrul de recepţie S0RBUF.

Şi recepţia datelor este dublu bufferată, aceasta permiţând recepţionarea caracterului următor înainte ca cel deja recepţionat să elibereze registrul S0RBUF.

Dacă sunt validate, indicatorul de eroare la încadrare şi indicatorul întrerupere la erori de recepţie S0EIR sunt setate dacă registrul S0RBUF nu a fost citit până la terminarea recepţiei caracterului următor.

Modul de lucru în buclă închisă (bitul S0LB) permite recepţionarea simultană a datelor transmise. De regulă, acest mod este folosit pentru testarea rutinelor specifice fără a fi necesare nici o legătură externă.

1.23.1 Modul asincron

În modul asincron, ASC0 permite o legătură full-duplex la care atât emiţătorul, cât şi receptorul folosesc acelaşi format al datelor şi aceiaşi viteză de transmisie.

Datele sunt transmise pe pinul TXD0 (P3.10) şi sunt recepţionate pe pinul RXD0 (P3.11).

Structura internă a modulului ASC0 în modul asincron este prezentată în figura 2.32.

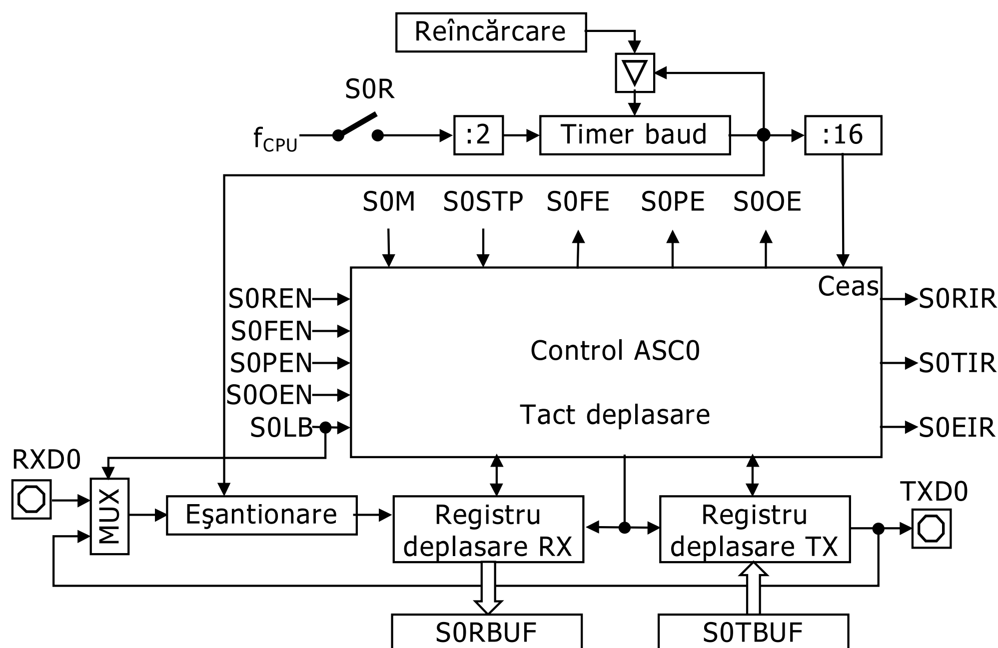


Figura 1.46. Structura ASC0 în modul asincron

Datele transmise în modul asincron pot avea următoarea structură:

- cadre de 7 biţi de date plus un bit de paritate generat automat (S0M=3h);
- cadre de 8 biţi de date (S0M=1h);
- cadre de 8 biţi de date plus un bit de paritate generat automat (S0M=7h);
- cadre de 9 biţi de date (S0M=4h).

Transmisia a 9 biţi permite şi folosirea unui regim special, de *atenţionare*, pentru facilitarea schimburilor de date pe o singură linie de transmisie în sistemele multiprocesor.

Astfel, dacă este selectat regimul atenţionare (S0M=5h), mesajele vor fi de două tipuri:

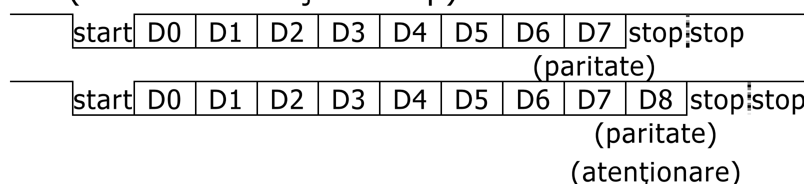
- adrese, când bitul 9 al mesajului este 1 LOGIC;
- date, când bitul 9 al mesajului este 0 LOGIC.

Când un procesor doreşte să transmită un bloc de date la alt procesor, mai întâi transmite un octet care identifică destinaţia. Un octet *adresă* este recepţionat de toate procesoarele conectate între ele şi dacă adresa este recunoscută de unul din ele, acesta comută modul atenţionare urmând să recepţioneze blocul de date care urmează.

Această procedură este direct utilizabilă în situaţia în care este un singur emiţător şi mai multe receptoare. Dacă se doreşte ca orice alt procesor să preia controlul asupra liniei de transmisie, este necesară adoptarea unei interfeţe care permite lucrul cu mai multe emiţătoare pe aceiaşi linie, de exemplu EIA RS-485.

Cadrele de date sunt formate din trei elemente de bază:

- bit de start;
- date (8 sau 9 biţi, mai întâi bitul mai puţin semnificativ);
- terminator (unu sau doi biţi de stop).



Transmisia asincronă porneşte la prima depăşire a timerului pentru generarea ratei de transmisie după încărcarea registrului S0TBUF. Înainte de transmiterea ultimului bit (un bit de stop), este setat indicatorul de întrerupere S0TBIR pentru semnaliza că datele au fost transmise.

Recepţia este declanşată de o tranziţie descrescătoare a semnalului de pe pinul RXD0. Datele recepţionate sunt eşantionate la o viteză de 16 ori mai mare decât viteza de transmisie, astfel încât valoarea bitului recepţionat este luată printr-o decizie majoritară relativ la eşantioanele 7, 8 şi 9. Dacă în urma deciziei majoritare bitul de start nu rezultă zero, operaţiunea de deserializare este stopată.

După ce ultimul bit a fost recepţionat, octetul este transferat în registrul S0RBUF şi este setat indicatorul S0RIR pentru a semnala recepţionarea unui octet.

1.23.2 Modul sincron

În modul sincron, ASC0 permite o legătură half-duplex utilizată de regulă, pentru extensii ale perifericelor de intrare-ieşire.

Datele sunt transmise şi recepţionate pe pinul RXD0 (P3.11) în timp ce pinul TXD0 (P3.10) este folosit pentru ceasul de transmisie.

Structura internă a modulului ASC0 în modul sincron este prezentată în figura 2.33.

Transmiterea sincronă începe la patru cicluri după ce datele au fost încărcate în S0TBUF. La transmiterea ultimului bit de date este setat indicatorul de întrerupere S0TBIR.

Recepţia sincronă este iniţiată la setarea bitului S0REN. Datele recepţionate sunt deplasate sincron cu tactul produs pe pinul TXD0. După al optulea tact, datele sunt transferate în S0RBUF şi este setat indicatorul de întrerupere S0RIR. Scrierea în registrul S0TBUF în timpul recepţiei nu are nici un efect asupra acesteia iar transmisia nu este iniţiată.

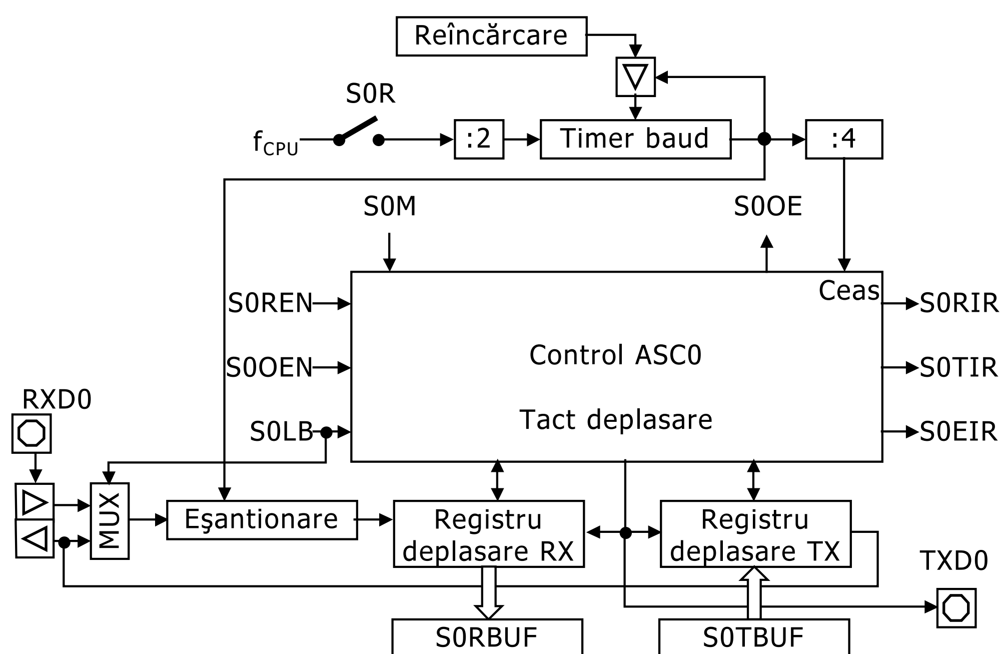


Figura 1.47. Structura ASC0 în modul sincron

Atenţie! Pentru transmisie pinul P3.11 (RXD0) trebuie setat ca ieşire (P3.11=1h şi DP3.11=1h).

Pentru recepţie, pinul P3.11 trebuie setat ca intrare (P3.11=1h şi DP3.11=0h).

În ambele cazuri, pinul P3.10 (TXD0) trebuie setat ca intrare (P3.10=1h şi DP3.10=0h).

1.23.3 Generarea ratei de transmisie

Blocul ASC0 are un timer de 13 biţi şi un registru de reîncărcare (S0BG) dedicat pentru generarea ratei de transmisie.

Timerul numără ceasul unităţii centrale divizat cu 2 şi poate fi comandat de bitul S0R. Fiecare depăşire a timerului produce un tact de deplasare pentru registrele de serializare. De asemenea, la depăşire timerul este încărcat cu valoarea din registrul S0BG.

Frecvenţa rezultată este divizată din nou, funcţie de modul de funcţionare selectat prin bitul S0BRS.

Modurile de determinare a vitezelor de transmisie asincrone, respectiv sincrone sunt prezentate în relaţiile următoare, iar câteva exemple de setări pentru vitezele standard pentru un procesor cu frecvenţa ceasului de 20 MHz în tabelul 2.49.

$$BAUD_{ASINC} = \frac{f_{CPU}}{(32 + 16 \cdot S0BRS) \cdot (1 + S0BG)}$$

$$BAUD_{SINC} = \frac{f_{CPU}}{(8 + 4 \cdot S0BRS) \cdot (1 + S0BG)}$$

Atenţie! Pentru obţinerea unor frecvenţe exacte, este posibilă utilizarea unui oscilator cu cuarţ special destinat (de exemplu f=18.432 MHz).

Tabelul 1.62

Viteză transmisie	S0BRS=0h		S0BRS=1h	
	Eroare	S0BG	Eroare	S0BG
625 kBaud	±0%	0000h	–	–
19.2 kBaud	-1.4% – +1.7%	001F–0020h	-1.4% – +3.3%	0014–0015h
9600 Baud	-1.4% – +0.2%	0040–0041h	-1.4% – +1.0%	002A–002Bh
4800 Baud	-0.6% – +0.2%	0081–0082h	-0.2% – +1.0%	0055–0056h
2400 Baud	-0.2% – +0.2%	0103–0104h	-0.2% – +0.4%	00AC–00ADh
1200 Baud	-0.4% – +0.2%	0207–0208h	-0.2% – +0.1%	015A–015Bh
600 Baud	-0.0% – +0.1%	0410–0411h	-0.1% – +0.1%	02B5–02B6h
75 Baud	+1.7%	1FFFh	-0.0% – +0.0%	15B2–15B3h

1.23.4 Controlul întreruperilor

Pentru operarea în condiţii normale, modulul ASC0 setează următorii indicatori de întrerupere:

- S0TBIR – activat când datele sunt mutate din S0TBUF în registrul de serializare pentru transmisie. Aparţine registrului S0TBIC şi generează întreruperea S0TBINT;
- S0TIR – activat înainte de transmiterea ultimului bit din cadrul transmis. Aparţine registrului S0TIC şi generează întreruperea S0TINT;
- S0RIR – activat când datele recepţionate sunt mutate în S0RBUF. Aparţine registrului S0RIC şi generează întreruperea S0RINT.

În cazul detectării unei erori (paritate, încadrare sau depăşire pentru modul asincron sau depăşire în modul sincron), este setat indicatorul corespunzător din registrul S0CON şi este declanşată o întrerupere S0EIR cu vectorul S0EINT.

Structura registrelor de control a întreruperilor S0TIC, S0TBIC, S0RIC şi S0EIC sunt prezentate în tabelul 2.17 din paragraful 1.16.1.

1.24. Interfaţa serială sincronă de viteză

Interfaţa serială sincronă de viteză (denumită în continuare SSC) permite un mod flexibil de comunicare între circuitul 80C167 şi alte microprocesoare, microcontrolere sau periferice externe.

Blocul SSC permite comunicaţii sincrone la viteze de până la 10 MBaud (pentru procesoare cu frecvenţa de ceas de 40 MHz).

Sunt programabile numărul de biţi, ordinea biţilor în mesaj, polaritatea şi faza tactului de serializare.

Registrele de transmisie şi recepţie sunt dublu bufferate. Generarea ratei de transfer este făcută prin intermediul unui timer propriu de 16 biţi.

Interfaţa SSC poate fi configurată în multe moduri, astfel încât ea poate fi compatibilă cu interfaţa ASC0 (în modul sincron), poate opera în sisteme multiprocesor sau poate realiza legătura cu periferice SPI.

Astfel, modulul SSC permite interfaţarea cu registre de deplasare externe (în scopul extinderii capabilităţilor de intrare-ieşire), memorii seriale (E^2ROM) sau alte controlere (de exemplu, pentru reţele locale).

Datele sunt transmise sau prelucrate pe pinul `MTSR` (`P3.9`) şi `MRST` (`P3.8`). Semnalul de sincronizare este obţinut pe pinul `SCLK` (`P3.13`).

Schema bloc a interfeţei SSC este prezentată în figura 2.34.

Figura 1.48. Structura interfeţei SSC

Registrul de control al interfeţei `SSCCON` are două moduri de utilizare, funcţie de starea bitului `SSCEN`, prezentate în tabelul 2.50:

- pe durata programării, `SSCEN=0h`, se pot modifica biţii de control (a);
- în utilizare, `SSCEN=1h`, se pot testa indicatorii de stare (b).

Tabelul 1.63

SSCCON (FFB0h) SSCEN=0	SSCEN	SSCMS	–	SSCAREN	SSCBEN	SSCPEN	SSCREN	SSCTEN	–	SSCPO	SSCPH	SSCHB	SSCBM			

SSCEN	Validare interfaţă SSC şi mod lucru SSCCON.
SSCMS	0h: circuit secundar; operează cu tact extern de la SCLK; 1h: circuit principal; generează semnal de tact pe SCLK.
SSCAREN	1h: SSC este resetat automat la o eroare a ratei de transmisie.
SSCBEN	1h: verifică erorile ratei de transmisie,
SSCPEN	1h: verifică erorile de fază.
SSCREN	1h: verifică erorile de la recepţie.
SSCTEN	1h: verifică erorile de la transmisie.
SSCPO	0h: semnalul SCLK inactiv este în starea 0 LOGIC; frontul activ este ↑; 1h: semnalul SCLK inactiv este în starea 1 LOGIC; frontul activ este ↓.
SSCPH	0h: deplasare date transmise pe front activ SCLK, încărcare date pe celălalt front; 1h: încărcare date recepţionate pe front activ SCLK, deplasare date pe celălalt front.

SSCHB	0h: transmite/recepţionează primul bitul mai puţin semnificativ; 1h: transmite/recepţionează primul bitul cel mai semnificativ.
SSCBM	Lungime mesaj: 1+SSCBM. 0h: rezervat.

SSCCON SSCEN=1	SSCEN	SSCMS	–	SSCBSY	SSCBE	SSCPE	SSCRE	SSCTE	–	–	–	–	SSCBM			

SSCEN	Validare interfaţă SSC şi mod lucru SSCCON.
SSCMS	Circuit secundar/principal.
SSCBSY	Setat cât timp o transmisie este în curs. Nu trebuie modificat.
SSCBE	Eroare transmisie (câtul între tactul existent şi cel presupus de circuitul secundar egal cu 2 sau 0.5).
SSCPE	Eroare fază (recepţionează schimbări ale datelor în zona frontului tactului de eşantionare).
SSCRE	Eroare recepţie (recepţie completă înainte ca bufferul să fie citit).
SSCTE	Eroare transmisie (este iniţiat un transfer înainte de modificarea registrului de transmisie)
SSCBM	Numărul bitul deplasat. Nu trebuie modificat.

Registrul de deplasare al SSC este conectat atât la pinul de emisie cât şi la pinul de recepţie. Astfel, transmisia şi recepţia datelor sunt sincronizate şi au loc în acelaşi timp.

La transmisie, ca circuit principal, procedura este iniţiată imediat după ce registrul SSCTB a fost încărcat. Ca circuit secundar, pornirea transmisiei este condiţionată de sosirea tactului de serializare. SSCTB este mutat în registrul de serializare imediat ce acesta este gol.

După ce transferul a început, este setat indicatorul de întrerupere SSCTIR pentru a semnala că este posibilă reîncărcarea registrului SSCTB. Când numărul programat de biţi (2…16) a fost transferat, conţinutul registrului de deplasare este mutat în registrul SSCRB şi va fi setat indicatorul de întrerupere SSCRIR.

Transferul datelor poate avea loc în mai multe moduri:

- numărul de biţi cuprins între 2 şi 16;
- transferul poate începe cu bitul cel mai semnificativ sau cel mai puţin semnificativ;
- tactul de deplasare în stare inactivă poate fi 0 LOGIC sau 1 LOGIC;
- biţii pot fi deplasaţi pe frontul activ sau inactiv al tactului de serializare;
- viteza de transmisie poate fi aleasă între 302 Baud şi 10 MBaud (pentru frecvenţa unităţii centrale de 40 MHz);
- tactul de deplasare poate fi generat (pentru circuitul principal) sau recepţionat (pentru circuitul secundar).

Alegând ordinea de transmisie a biţilor (SSCHB=0h), interfaţa este compatibilă cu interfaţa ASC0 a circuitului 80C167 sau cu interfaţa serială a familiei 8051; selectând SSCHB=1h, comunicaţia este compatibilă cu perifericele SPI. Indiferent de ordinea de transmitere a biţilor, în registrele SSCTB şi SSCRB datele vor fi întotdeauna aliniate la dreapta, cu bitul cel mai puţin semnificativ pe poziţia 0.

Interfaţa SSC foloseşte trei linii a portului P3 pentru legătura cu celelalte circuite. Setările acestor pini depind de modul de operare, circuit principal sau secundar.

Direcţia pinilor (intrare-ieşire) depinde de modul de operare. Pentru a funcţiona ca ieşire, este obligatorie setarea bistabilul de ieşire respectiv întrucât ieşirea pinului este produsă de o poartă ŞI-LOGIC care admite ca intrări bistabilul de ieşire (setat, 1 LOGIC) şi funcţia alternativă a portului.

Direcţia pinilor trebuie selectată de utilizator funcţie de modul de lucru, conform cu tabelul 2.51.

Tabelul 1.64

Pin	Circuit principal			Circuit secundar			
	Funcţie	Bistabil	Direcţie	Funcţie	Bistabil	Direcţie	Tip
SCLK (P3.13)	ieşire	P3.13=1	DP3.13=1	Intrare	P3.13=x	P3.13=0	–
MTSR (P3.9)	ieşire	P3.9=1	DP3.9=1	Intrare	P3.9= x	DP3.9=0	–
MRST (P3.8)	intrare	P3.8=x	DP3.8=0	Ieşire	P3.8=1	DP3.8=1	ODP3.8=1

1.24.1 Operarea full-duplex

În acest mod, dispozitivele sunt legate pe trei linii. Definirea acestor linii este atributul circuitului principal: linia conectată la pinul MTSR (*master transmit slave receive*) este pentru transmisie iar linia conectată la pinul MRST (*master receive slave transmit*) este pentru recepţie.

Numai circuitul principal, care poate fi unul singur la un moment dat, transmite tactul de serializare pe pinul SCLK; toate circuitele secundare primesc acest tact, aşa că pinul corespunzător trebuie setat ca intrare (DP3.13=0h).

Este obligatoriu ca, la iniţializare, să se stabilească circuitul principal, descrierea modului de operare pentru fiecare bloc SSC (registrul SSCCON) iar pentru toate circuitele din sistem să se facă selectările necesare pentru caracteristicile pinilor (intrări sau ieşiri, tipuri de ieşiri etc.).

Pinii de ieşire MRST ai circuitelor secundare sunt legaţi împreună pe un singur fir. Există două posibilităţi de a evita coliziunea datelor:

- stabilirea prin program, a unui singur circuit secundar cu pinul MRST activ la un moment dat. De exemplu, toate circuitele secundare au dezactivată transmisia, circuitul principal emite o cerere de date de la un anume circuit secundar care îşi activează linia MRST, transmite datele, după care dezactivează din nou linia.
- setarea liniilor MRST ca ieşiri cu drenă în gol, fiind posibilă astfel realizarea unei conexiuni ŞI-CABLAT. Şi în acest caz identificarea circuitului secundar care a transmis mesajul se poate face fie prin transmiterea unei adrese de identificare de către circuitul principal, fie prin selectarea hardware a perifericului dorit.

Iniţializarea pinului SCLK a circuitului principal trebuie făcută cu atenţie pentru a nu genera impulsuri parazite care să perturbe circuitele secundare. Procedura recomandată este următoarea:

- setare nivel inactiv linie SCLK – SSCPO='x';

- setare pin port cu valoarea anterioară – P3.13='x';
- setare pin ca ieşire – DP3.13=1h;
- validare interfaţă SSC – SSCEN=1h;
- dacă SSCPO=0h, în final se setează bistabilul de ieşire – P3.13=1h.

În acest mod de funcţionare este posibilă schimbarea rolurilor între circuitul principal şi unul secundar. Această modificare trebuie făcută cu atenţie la ordinea reprogramării pinilor, pentru a nu provoca conflicte de magistrală.

Structura unui sistem full-duplex este prezentată în figura 2.35.

1.24.2 Operarea half-duplex

În modul de lucru half-duplex este necesară o singură linie atât pentru transmisia cât şi pentru recepţia datelor. Datele sunt schimbate pe linia care conectează pinii MRST şi MTSR ale tuturor dispozitivelor, în timp ce semnalul de sincronizare este asigurat de legătura între pinii SCLK.

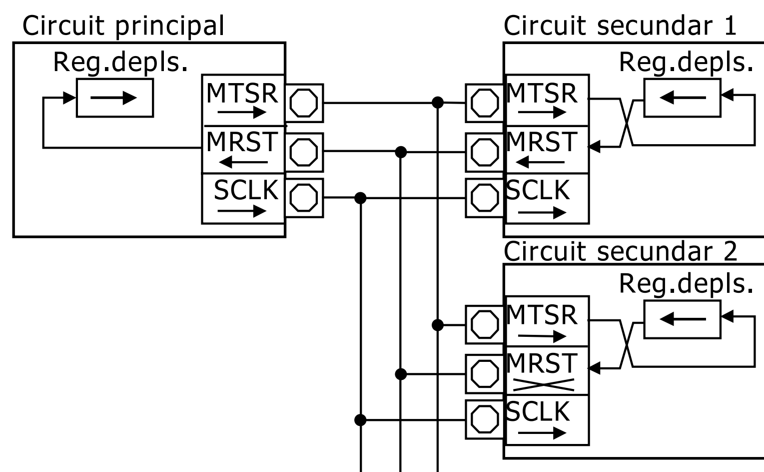


Figura 1.49. Interfaţa SSC în modul full-duplex

Dispozitivul principal controlează transferul datelor prin generarea tactului de transmisie iar circuitul secundar îl recepţionează. Deoarece schimbul de date se face pe o singură linie, transmisia între expeditor şi destinatar în situaţia în care aceştia se află la distanţă mare se poate face şi indirect, datele fiind schimbate între dispozitive intermediare.

Asemănător cu modul full-duplex, există două metode pentru a evita coliziunea datelor:

- un singur transmiţător este activ la un moment dat pe linie;
- dispozitivele care nu transmit date au selectate ieşiri de tip drenă în gol.

Deoarece pinii de ieşire şi de intrare (MTSR şi MRST) sunt conectaţi împreună la fiecare circuit, un circuit emiţător va regăsi informaţia transmisă şi la receptorul propriu, prin compararea datelor transmise şi recepţionate fiind astfel posibilă detectarea rapidă a unor probleme pe linia de transmise (scurtcircuit, linii neadaptate etc.).

Structura unui sistem half-duplex este asemănătoare cu structura sistemului full-duplex prezentată în figura 2.33, singura diferenţă fiind dată

de dispariţia legăturilor de date MTSR→MTSR→…, MRST→MRST→… care în această arhitectură devine MTSR+MRST→MTSR+MRST→….

1.24.3 Viteza de transmisie

Interfaţa serială SSC are propriul generator de rate de transmisie, cu timer de 16 biţi şi cu facilitatea de reîncărcare.

Generatorul are asigurată frecvenţa de intrare dintr-un divizor cu 2 a frecvenţei unităţii centrale. Timerul numără continuu descrescător şi poate fi validat prin bitul SSCEN din registrul SSCCON.

Registrul de reîncărcare SSCBR are două funcţiuni: la citire, cu interfaţa SSC validată, restituie conţinutul timerului; dacă interfaţa SSC este dezactivată, indică valoarea de reîncărcare.

Formula pentru determinarea vitezei de transmisie este prezentată în relaţia următoare:

$$BAUD_{SSC} = \frac{f_{CPU}}{2 \cdot (1 + SSCBR)}$$

Câteva valori orientative ale vitezei de transmisie pentru un procesor cu frecvenţa ceasului de 20 MHz sunt prezentate în tabelul 2.52.

Tabelul 1.65

Viteză transmisie	Valoare reîncărcare	Viteză transmisie	Valoare reîncărcare
REZERVAT	0000h	1 MBaud	0009h
5 MBaud	0001h	100 kBaud	0063h
3.3 MBaud	0002h	10 kBaud	03E7h
2.5 MBaud	0003h	1 kBaud	270Fh
2 MBaud	0004h	152.6 Baud	FFFFh

1.24.4 Detectarea erorilor

Interfaţa SSC este capabilă să detecteze patru tipuri de erori:

- Erori de recepţie, pentru circuitul principal sau secundar, când un nou mesaj a fost complet recepţionat iar datele anterior recepţionate nu au fost descărcate din registrul SSCRB. Este setat indicatorul SSCRE validat de bitul SSCREN, ambele din registrul SSCCON.
- Erori de fază, când datele de pe pinul MRST (al circuitului principal) sau pinul MTSR (al circuitului secundar) îşi schimbă starea în zona frontului activ al semnalului SCLK (în intervalul unui eşantion înainte, respectiv două eşantioane după frontul activ). Este setat indicatorul SSCPE validat de bitul SSCPEN, ambele din registrul SSCCON.
- Erori de rată de transmisie, numai pentru circuitul secundar, dacă semnalul SCLK are o deviaţie mai mare de 100% faţă de valoarea programată. Pentru a fi detectată această eroare, este necesar ca şi circuitul secundar să aibă programată o viteză de transmisie cu circuitul principal. Este setat indicatorul SSCBE validat de bitul SSCBEN, ambele din registrul SSCCON.
- Erori de transmisie, numai pentru circuitul secundar, în momentul în care un circuit principal solicită date de la un circuit secundar iar acesta nu a schimbat datele din registrul SSCTB. În această situaţie, circuitul

secundar va emite totuşi vechiul conţinut al registrului de transmisie dar este posibilă coliziunea datelor, în sensul că linia de transmisie poate fi deja ocupată de alt emiţător. Este setat indicatorul SSCTE validat de bitul SSCTEN, ambele din registrul SSCCON.

1.24.5 Controlul întreruperilor SSC

Întreruperile generate de interfaţa SSC sunt gestionate de trei registre de întrerupere SSCRIC, SSCTIC şi SSCEIC respectiv pentru evenimente de recepţie, transmisie şi erori.

Întreruperea de recepţie este generată în momentul în care este copiat registrul de deplasare SSC în registrul SSCRB. Este semnalată de indicatorul SSCRIR şi are vectorul SCRINT.

Întreruperea de transmisie este generată în momentul în care este copiat registrul SSCTB în registrul de deplasare SSC şi, în cazul circuitelor secundare, transmisia a început prin recepţionarea semnalului SCLK. Este semnalată de indicatorul SSCTIR şi are vectorul SCTINT.

Întreruperea generată de erori a fost prezentată în paragraful 1.24.4, este semnalată de indicatorul SSCEIR şi are vectorul SCEINT.

Structura registrelor de control a întreruperilor SSCRIC, SSCTIC şi SSCEIC sunt prezentate în tabelul 2.17 din paragraful 1.16.1.

1.25. Încărcătorul bootstrap

Încărcătorul bootstrap, denumit în continuare BSL, asigură un mecanism pentru încărcarea programului de lucru la microcontrolerului prin intermediul interfeţei seriale ASC0, fără a fi necesară existenţa nici unui circuit ROM (fie el intern sau extern).

Modulul BSL încarcă datele în memoria RAM internă, dar este posibilă şi transferarea programului în memoria externă, folosind o rutină secundară.

Blocul BSL poate fi folosit pentru încărcarea aplicaţiei complete în sistemele fără circuite ROM, poate încărca numai programe temporare, de test sau calibrare în sistemele definitivate ori se poate utiliza pentru încărcarea programelor pentru dispozitivele Flash-ROM.

1.25.1 Intrarea în modul BSL

Familia 80C16x intră în acest mod dacă, la sfârşitul iniţializării, pinul P0L.4 are nivel 0 LOGIC. Codurile pentru execuţia BSL sunt memorate într-un circuit intern special, Boot-ROM, care nu are nici o legătură cu memoria ROM internă.

După intrarea în acest mod, unitatea centrală explorează pinul RXD0 pentru a detecta un octet zero (adică un bit de start, opt biţi 0h şi un bit de stop). Funcţie de durata de recepţie a acestui octet, unitatea centrală determină viteza de transmisie şi iniţializează interfaţa ASC0 în mod corespunzător. Folosind aceiaşi viteză de transmisie, circuitul răspunde pe

interfaţă cu un octet de control funcţie de tipul său: 80C165–AAh, 80C166-55h, 80C167–A5h.

În momentul în care microcontrolerul întră în modul BSL, registrele de control sun setate automat la următoarele valori:

- Timer WDT – dezactivat;
- Indicator context CP – FA00h;
- Indicator stivă SP – FA40h
- Registru STKUN – FA40h;
- Registru STKOV – FA0Ch;
- Registru SYSCON – FA40h;
- Registru S0CON – 8011h;
- Registru S0BG – funcţie de viteză;
- Registru BUSCON0 – funcţie de configuraţia de iniţializare;
- TXD0 (P3.10 şi DP3.10) – 1h.

Atenţie! Chiar dacă memoria ROM internă este validată prin configuraţia de iniţializare, nu se va executa nici o instrucţiune din ea.

Spre deosebire de iniţializarea normală, timerul WDT este dezactivat astfel încât secvenţa de încărcare BSL nu este limitată ca timp.

Sistemele care utilizează procedura BSL o pot iniţia prin conectarea unei rezistenţe de 5 kΩ între pinul P0L.4 şi masă. Dacă sistemul foloseşte temporar modul BSL se poate adapta o soluţie cu un jumper sau un semnal extern.

1.25.2 Procedura de lucru BSL

După trimiterea octetului de identificare, blocul BSL întră într-o buclă aşteptând 32 de octeţi pe interfaţa ASC0. Aceşti octeţi sunt memoraţi consecutiv în memoria RAM internă, de la adresa 00’FA40h până la adresa 00’FA5Fh, adică 16 instrucţiuni. Unitatea centrală execută un salt la prima instrucţiune şi, astfel, secvenţa BSL se consideră încheiată.

Totuşi, o aplicaţie nu poate avea numai 16 octeţi, aşa că se intră în a doua buclă de încărcare. Aceasta, folosind deja parametrii setaţi pentru interfaţa ASC0, poate continua încărcarea, în orice zonă de memorie RAM a programului de aplicaţie, rutine diverse, blocuri de date etc.

Procesul poate continua în mai mulţi paşi sau poate trece direct la execuţia programului încărcat dar, în toate situaţiile, unitatea centrală rămâne în modul BSL.

Pentru a executa aplicaţia în modul normal de lucru, este necesară ieşirea din modul BSL. Aceasta se poate face prin program, executând o instrucţiune de iniţializare software – SRST, care nu mai verifică starea pinului P0L.4 sau printr-o iniţializare externă, caz în care pinul P0L.4 trebuie să fie în starea 1 LOGIC.

Calcularea vitezei de transfer este făcută prin intermediul timerului T6 care măsoară durata octetului nul iniţial. Orice eroare de măsurare a acestei durate poate conduce la deviaţii mari de viteză între rata de transfer reală şi rata de transfer calculată de circuit. Pentru un transfer corect de date, este obligatoriu ca această deviaţie să fie mai mică de 2.5%. În principiu, cu cât

viteza de transfer a datelor este mai mică, cu atât eroarea de corelare a vitezelor este mai mică.

1.26. Consumul redus de energie

Pentru a mări durata de funcţionare a unui sistem care are alimentarea asigurată de la baterii sau acumulatori cu o resursă limitată, circuitul 80C167 are prevăzute două regimuri economice de funcţionare *inactiv* şi *oprit*.

1.26.1 Modul inactiv

În acest mod, toate perifericele, inclusiv timerul WDT sunt funcţionale; este oprită numai funcţionarea unităţii centrale.

Coeficientul de reducere al puterii consumate depinde de frecvenţa procesorului şi este dată de relaţia următoare:

$$K = \frac{20 + 3f_{CPU}[MHz]}{30 + 8f_{CPU}[MHz]} \approx 0.4$$

Intrarea în modul inactiv este făcută prin executarea instrucţiunii IDLE.

Terminarea modul inactiv se face prin orice cerere de întrerupere care are indicatorul xxxIE validat. După executarea instrucţiunii RETI de la sfârşitul rutinei de tratare a întreruperii, unitatea centrală continuă execuţia programului de la următoarea instrucţiune după IDLE. Pentru o cerere care este programată pentru un serviciu PEC, după terminarea transferului unitatea centrală rămâne în modul IDLE.

Atenţie! Întreruperile sau serviciile PEC vor fi executate şi vor scoate procesorul din modul inactiv numai dacă prioritatea lor curentă este mai mare decât a unităţii centrale (stabilită în registrul PSW – câmpul ILVL).

O altă cale de terminare a modului inactiv este activarea semnalului extern de întrerupere nemascabilă $\overline{NMI}$ ori executarea unei iniţializări externe.

De asemenea, este posibilă ieşirea din modul inactiv printr-o iniţializare produsă de timerul WDT.

1.26.2 Modul oprit

Pentru a reduce şi mai mult consumul de energie, programatorul are la dispoziţie şi acest regim. În modul oprit toată activitatea microcontrolerului este stopată, numai conţinutul memoriei RAM păstrându-se nealterat.

Coeficientul de reducere al puterii consumate este semnificativ, în plus faţă de modul inactiv fiind posibilă şi reducerea tensiunii de alimentare cu 50%.

Modul oprit asigură reducerea puterii consumate de circa 20000 de ori, prelungindu-se corespunzător durata de funcţionare a sursei de alimentare.

Intrarea în modul oprit este făcută prin intermediul instrucţiunii PWRDN.

Modul oprit poate fi terminat numai prin generarea unei iniţializări externe.

Pentru a proteja unitatea centrală de o intrare neintenţionată în modul oprit, există două modalităţi:

- instrucţiunea PWRDN este o instrucţiune protejată, pe 32 de biţi, fiind puţin probabil să fie executată din întâmplare;
- instrucţiunea este efectivă numai dacă semnalul $\overline{NMI}$ este activ în timpul executării instrucţiunii.

Ultima facilitate este utilă în conjuncţie cu un semnal extern pentru semnalarea unei căderi a tensiunii de alimentare, cuplat pe pinul $\overline{NMI}$. Rutina de tratare a întreruperii $\overline{NMI}$ salvează starea unităţii centrale în stivă iar la final execută instrucţiunea PWRDN. Dacă pinul $\overline{NMI}$ este încă în 0 LOGIC datorită alimentării, procesorul va intra în modul oprit; dacă nu va funcţiona normal în continuare.

1.26.3 Starea pinilor de ieşire pe parcursul modurilor economice

Pe durata modului inactiv, când oscilatorul unităţii centrale este oprit, toate perifericele îşi continuă funcţionarea lor normală. Din acest motiv, toate porturile configurate ca ieşiri păstrează ultima valoare scrisă în bistabilii de ieşire.

Dacă pinul este folosit de un modul intern, starea sa reflectă situaţia perifericului.

Pinii care sunt folosiţi pentru controlul magistralei externe trec într-o stare care reprezintă valoarea lor inactivă (de exemplu, $\overline{WR}$ trece în 1 LOGIC iar ALE în 0 LOGIC) iar alţii au valoarea de la ultimul acces EBC (de exemplu $\overline{BHE}$).

Dacă magistrala externă este în mod multiplexat cu date pe 8 biţi, portul P0H redă ultima adresă folosită de EBC iar, în caz contrar, este în înaltă impedanţă. Portul P0L este întotdeauna în stare de înaltă impedanţă în modul inactiv.

Pe portul P1, pentru magistrală externă demultiplexată, se regăseşte ultima adresă utilizată.

Portul P4 conţine adresa segment a ultimului acces EBC. Pinii nefolosiţi pentru adresă sunt pini de intrare-ieşire şi, în consecinţă, redau ultima valoare scrisă.

În modul oprit, atât pentru unitatea centrală cât şi pentru modulele interne oscilatorul este blocat. Ca în modul inactiv, toate porturile de ieşire reflectă starea înainte de intrarea în acest mod. Dacă pinul este folosit de un periferic intern, starea sa redă ultima acţiune executată de periferic.

1.27. Setul de instrucţiuni

Pentru a obţine performanţe maxime într-o arhitectură cu stivă de instrucţiuni, setul de instrucţiuni a fost optimizat pentru a funcţiona în filozofia unui procesor RISC. Această filozofie a condus la următoarele tendinţe pentru realizarea instrucţiunilor:

- Coduri realizarea unor operaţiuni care necesită secvenţe de instrucţiuni utilizate frecvent. Se evită transferul în şi din registre temporare, cum ar fi acumulatorul sau biţi de tip transport sau împrumut. Este permisă executarea în paralel a sarcinilor, de exemplu salvarea stării procesorului la intrarea în întreruperi sau subrutine.
- Este evitată o structură complicată de codificare şi, de asemenea, structuri complexe de adresare. Aceasta scade timpul decodificării instrucţiunilor permiţând şi o dezvoltare simplă a compilatoarelor.
- Cele mai utilizate instrucţiuni sunt codificate într-un singur cuvânt. Aceasta permite alinierea codului la nivel de cuvânt şi evitarea existenţei unor circuite complicate pentru aliniere.

Operanzii admişi de procesor sunt de tip cuvânt (16 biţi), octet (8 biţi) sau bit. Modurile de bază de adresare folosite pentru operanzi sunt directe, indirecte sau imediate.

În cele ce urmează sunt prezentate instrucţiunile circuitului 80C167 organizate pe clase şi pe tipuri de operanzi.

Instrucţiuni aritmetice

◦ Adunarea a două cuvinte sau octeţi	ADD	ADDB
◦ Adunarea cu transport a două cuvinte sau octeţi	ADDC	ADDCB
◦ Scăderea a două cuvinte sau octeţi	SUB	SUBB
◦ Scăderea cu transport a două cuvinte sau octeţi	SUBC	SUBCB
◦ Înmulţire 16·16 biţi cu sau fără semn	MUL	MULU
◦ Împărţire 16·16 biţi cu sau fără semn	DIV	DIVU
◦ Împărţire 32·16 biţi cu sau fără semn	DIVL	DIVLU
◦ Complement faţă de 1 pentru cuvânt sau octet	CPL	CPLB
◦ Complement faţă de 2 pentru cuvânt sau octet	NEG	NEGB

Instrucţiuni logice

◦ ŞI-LOGIC între două cuvinte sau octeţi	AND	ANDB
◦ SAU-LOGIC între două cuvinte sau octeţi	OR	ORB
◦ SAU-EXCLUSIV între două cuvinte sau octeţi	XOR	XORB

Comparări şi control bucle

◦ Comparare între două cuvinte sau octeţi	CMP	CMPB
◦ Comparare între două cuvinte cu postincrementare cu 1 sau 2	CMPI1	CMPI2
◦ Comparare între două cuvinte cu postdecrementare cu 1 sau 2	CMPD1	CMPD2

Instrucţiuni booleene biţi

◦ Manipulare câmp de biţi în octet superior sau inferior	BFLDH	BFLDL
◦ Setare bit	BSET	

- Ştergere bit BCLR
- Mutare bit BMOV
- Mutare bit negat BMOVN
- ŞI-LOGIC între doi biţi BAND
- SAU-LOGIC între doi biţi BOR
- SAU-EXCLUSIV între doi biţi BXOR
- Comparare între doi biţi BCMP

Instrucţiuni deplasare şi rotire

- Deplasare dreapta cuvânt SHR
- Deplasare stânga cuvânt SHL
- Rotire dreapta cuvânt ROR
- Rotire stânga cuvânt ROL
- Deplasare aritmetică dreapta cuvânt ASHR

Instrucţiuni normalizare

- Determinare număr deplasări normalizare cuvânt PRIOR

Instrucţiuni mutare date

- Mutare standard cuvânt sau octet MOV MOVB
- Mutare octet la o locaţie de cuvânt cu extensie de semn sau de zero MOVBS MOVBZ

Instrucţiuni stiva sistem

- Introducerea unui cuvânt în stivă PUSH
- Extragerea unui cuvânt din stivă POP
- Schimbarea contextului registrelor GPR SCXT

Instrucţiuni salt

- Salt condiţionat la o adresă absolută, indirectă sau relativă în segmentul curent JMPA JMPI JMPR
- Salt necondiţionat la adresă în orice segment JMPS
- Salt condiţionat la o adresă relativă în segmentul curent funcţie de starea unui bit JB JNB
- Salt condiţionat la o adresă relativă în segmentul curent funcţie de starea unui bit cu inversarea bitului în caz de salt JBC JNBS

Instrucţiuni apel subrutine

- Apel condiţionat la adresă absolută, indirectă sau relativă în segmentul curent CALLA CALLI CALLR
- Apel necondiţionat la adresă în orice segment CALLS
- Apel condiţionat la o adresă absolută în segmentul

curent şi salvarea în stivă a unui registru selectabil PCALL
- Apel necondiţionat de întrerupere sau excepţie TRAP

Instrucţiuni reîntoarcere subrutine

- Reîntoarcere din segmentul curent RET
- Reîntoarcere din orice segment RETS
- Reîntoarcere din segmentul curent şi aducerea din stivă a unui registru selectabil RETP
- Reîntoarcere din rutină întrerupere RETI

Instrucţiuni control sistem

- Iniţializare software SRST
- Intrare mod inactiv IDLE
- Intrare mod oprit PWRDN
- Servire timer WDT SRVWDT
- Invalidare timer WDT DISWDT
- Terminare rutină iniţializare EINIT

Instrucţiuni diverse

- Nici o acţiune NOP
- Definirea unor instrucţiuni neintreruptibile ATOMIC
- Comutare moduri adresare 'reg', 'bitoff' şi 'bittadr' pentru zona ESFR EXTR
- Suprapunere pagini date folosind o pagină specială de date în locul DPP şi opţional comutare în zona ESFR EXTP EXTPR
- Suprapunere pagini date folosind un segment specific în locul DPP şi opţional comutare în zona ESFR EXTS EXTSR

Unele instrucţiuni care sunt critice pentru funcţionare unităţii centrale sunt denumite *instrucţiuni protejate.* Pentru a creşte protecţia împotriva unor încărcări eronate de cod, aceste instrucţiuni folosesc 32 de biţi pentru decodificare. O instrucţiune protejată trebuie să aibă codul operaţiunii repetat de două ori în al doilea cuvânt al instrucţiunii iar octetul următor codului să fie complementul acestuia.

Instrucţiunile protejate sunt DISWDT, EINIT, IDLE, PWRDN, SRST şi SRVWDT. Apariţia unei erori de decodificare la o astfel de instrucţiune provoacă apariţia unei excepţii hardware prin setarea indicatorului PRTFLT din registrul TFR (prezentate în 1.16.7).

Atenţie! Circuitul nu dispune de aritmetică BCD. Calculele BCD pot fi efectuate prin convertirea datelor BCD în cod hexazecimal, efectuarea calculelor, urmată de convertirea rezultatului în BCD. Se pot folosi facilităţile de înmulţire/împărţire ale circuitului sau, dacă

există suficient spaţiu, se pot face conversii rapide pe bază de tabele.

Instrucţiunea PRIOR permite emularea simplă a operaţiilor în virgulă flotantă cu un număr mic de alte instrucţiuni.

Dacă este folosită pentru iniţializare memoria ROM internă (pinul EA=1h la iniţializare), circuitul trebuie să conţină această memorie şi trebuie să aibă un vector de iniţializare valid şi un program corespunzător.

La alocarea memoriei ROM interne în segmentul 0 sau 1 nu trebuie executate instrucţiuni decât din memoria externă sau RAM intern.

2. Dezvoltarea sistemelor cu microcontrolere

Scopul acestui capitol este de a prezenta câteva unelte software strict necesare oricărui utilizator, sisteme de dezvoltare pentru testarea soluţiilor hardware şi software şi nu în ultimul rând, câteva aplicaţii şi drivere de uz general cu un accent deosebit pus pe interfeţele I^2C.

Având ca punct de pornire aceste cunoştinţe, un utilizator îşi poate dezvolta propriile sale aplicaţii, în cele mai diverse domenii: sisteme de alarmă, automatizări casnice din cele mai diverse, îmbunătăţiri ale electronicii automobilului etc. Practic, domeniul de utilizare al microcontrolerelor nu este limitat decât de natura semnalelor care trebuiesc controlate (şi, evident, de existenţa traductoarelor necesare) şi de imaginaţia utilizatorului.

2.1. Software

Programele aplicative sunt scrise, de regulă, în asamblor sau în C. Există multe alte programe ajutătoare, compilatoare, emulatoare create de diverse firme. Dintre acestea se pot evidenţia:

- ApBUILDER de la firma Intel care, prin intermediul unor ferestre, poate seta parametrii blocurilor funcţionale ale microntrolerelor generând rutinele aferente în asamblare sau în C. Versiunea 2.21 a programul suportă următoarele tipuri de microcontrolere şi microprocesoare:
  - Intel386^{TM} EX;
  - 80C186EA / 80C188EA, 80C186EB / 80C188EB, 80C186EC / 80C188EC, 80C186XL / 80C188XL;
  - 8XC196KD, 8XC196KC, 8XC196KB, 8XC198, 8XC196KR, 8XC196KQ, 8XC196KT, 8XC196JR, 8XC196JQ 8XC196JT, 8XC196NP, 8XC196NT, 8XC196NU;
  - 80C296SA;
  - 8XC52, 8XC54, 8XC58, 8XC51FA, 8XC51FB, 8XC51FC;
  - 8XC251SA, 8XC251SB, 8XC251SP, 8XC251SQ;
  - 8XC151SA, 8XC151SB;
  - 8X930Ax USB;
  - 8X930Hx.

Detalii suplimentare se pot găsi la pagina http://www.intel.com sau http://developer.intel.com/design/mcs51/.

Franklin Software (http://www.fsinc.com) a realizat unul din cele mai bune compilatoare C pentru familia de microcontrolere 80x51 şi 80xC552. De asemenea, Franklin Software a dezvoltat întreaga gamă de aplicaţii software (asamblor, compilator C, emulator, monitor etc.) pentru microcontrolerele de 16 biţi din familia Siemens 80C16X.

La Philips (http://www.semiconductors.philips.com) se poate găsi un asamblor şi un depanator pentru familiile 80xC51 şi XA cât şi referinţe la alte programe realizate de alte firme.

2.1.1 Compilatorul C

Limbajul C folosit este bazat pe ANSI C cu câteva modificări datorate:

- împărţirii spaţiului de adresare al unităţii centrale;
- utilizarea specifică a registrelor speciale (SFR);
- variabilelor de tip bit şi obiectele adresabile la nivel de bit;
- opţiunile bancurilor de registre şi mascarea registrelor de uz general;
- întreruperi specifice pentru familiile de microcontrolere.

Vor fi prezentate în continuare principalele caracteristici ale compilatoarelor C51 pentru 8xC552, respectiv C166 pentru 80C16x. Compilatoarele nu sunt universale, ele fiind specifice fiecărei familii de microcontrolere, generând un cod compact şi extrem de rapid. Pentru informaţii suplimentare se recomandă lucrările *C51 COMPILER User's Guide, Keil Elektronik GmbH* şi *C166 COMPILER User's Guide, Keil Elektronik GmbH*.

Folosirea limbajului de nivel înalt oferă câteva avantaje:

- nu este necesară cunoaşterea setului de instrucţiuni ale procesoarelor, însă este de dorit (dar nu absolut necesar) cunoaşterea structurii memoriei unităţilor centrale;
- detaliile referitoare la alocarea registrelor şi a diferitelor zone de memorie cad în sarcina compilatorului;
- programul are o formă structurată;
- programarea şi timpul de testare a programelor este redus drastic, mărind eficienţa;
- librăriile C suportate conţin multe rutine standard, cum ar fi: intrări/ieşiri formatate, conversii numerice, aritmetică în virgulă flotantă;
- utilizatorul îşi poate crea propriile librării;
- compilatoarele C realizate sunt portabile, permiţând trecerea rapidă de la un tip de procesor la altul.

Sintaxa pentru invocarea compilatoarelor este următoarea:

C51 `fişier.c [listă_control]`

C166 `fişier.c [listă_control]`

unde: `fişier.c` reprezintă programul sursă care va fi compilat şi transformat în fişier obiect *.obj*.

`listă_control` conţine directivele din linia de apel.

Cele mai importante comenzi din `listă_control` utilizate de şi C51 C166 sunt prezentate în continuare:

- `TINY` (numai C166), `SMALL`, `COMPACT`, `MEDIUM` (numai C166) sau `LARGE` selectează modelul de memorie corespunzător;
- `CODE` adaugă la fişierul de ieşire *.lst* mnemonicele în asamblor generate de compilator;
- `DEBUG` adaugă la fişierul de ieşire *.obj* informaţii pentru depanare;
- `HOLD` (numai C166) specifică explicit spaţiul de memorie utilizat de obiecte `near`, `idata`, `sdata` sau `bdata`;

Exemplu: `C166 prog.c HOLD (near 6)` – toate variabilele care ocupă mai puţin de 6 octeţi sunt alocate în zona `near`.

- `INTERVAL` (numai C51) specifică un interval diferit de 3 octeţi pentru vectorii de întrerupere;
- `INTVECTOR` (numai C51) declară deplasamentul pentru tabela vectorilor de întrerupere în situaţia în care tabela nu este memorată de la adresa 0h;
- `MOD167` validează utilizarea instrucţiunilor 80C167;
- `OPTIMIZE` inspectează codul final eliminând unele slăbiciuni ale compilatoarelor. `OPTIMIZE` este urmat de o valoare zecimală care descrie tipul de optimizare ales:
  - `(0)` îmbunătăţeşte timpul de prelucrare prin efectuarea calculelor care conţin constante, inclusiv calculul adreselor; de asemenea, compilatorul elimină salturile inutile (`JMP` la `JMP`);
  - `(1)` elimină fragmentele de cod neutilizate şi analizează salturile condiţionate pentru a sesiza dacă prin schimbarea condiţiei acesta poate fi eliminat;
  - `(2)` ameliorează accesul la variabilele care sunt incluse direct în operatori, eliminând necesitatea utilizării de registre intermediare;
  - `(3)` elimină instrucţiunile `MOV` redundante;
  - `(4)` nu mai rezervă memorie pentru variabilele şi parametrii care sunt transferaţi prin registre; de asemenea, optimizează instrucţiunile `CASE/SWITCH` prin transformarea lor în tabele de salturi sau în secvenţe comparare-salt;
  - `(5)` permite calculul unic a unor expresii identice care apar în rutine; de asemenea, optimizează buclele din program, în sensul încărcării valorilor constante în afara buclei;
- `PECDEF` (numai C166) rezervă spaţiul necesar pentru pointerii sursă şi destinaţie ai PEC;
- `SRC` creează în locul fişierului obiect un fişier sursă care poate fi asamblat cu macroasamblorul corespunzător.

Tipuri de date

În tabelul 3.1.a) sunt prezentate cuvintele cheie suplimentare pentru C51 iar în tabelul 3.1.b) pentru C166. Suplimentar, variabilele pot fi combinate în structuri sau uniuni, şiruri multidimensionale iar adresarea lor poate fi făcută prin pointeri. Directiva de tip `sfr`, atât pentru C51 cât şi pentru C166 simplifică adresarea registrelor speciale.

Tabelul 2.1

Tip date	Dimensiune	Domeniu de valori
a) C51		
bit	1 bit	0 sau 1
signed char	1 octet	-128 ÷ +127
unsigned char	1 octet	0 ÷ +255

signed int	2 octeţi	-32768 ÷ +32767
unsigned int	2 octeţi	0 ÷ +65535
signed long	4 octeţi	-2147483648 ÷ +2147483647
unsigned long	4 octeţi	0 ÷ +4294967295
float	4 octeţi	±1.176E-38 ÷ ±3.40E+38
pointer	1÷3 octeţi	adresa obiectului în funcţie de zona de memorie
Tipuri de date pentru SFR		
sbit	1 bit	0 sau 1
sfr	1 octet	0 ÷ +255
sfr16	2 octeţi	0 ÷ +65535
b) C166		
bit	1 bit	0 sau 1
signed char	1 octet	-128 ÷ +127
unsigned char	1 octet	0 ÷ +255
signed int	2 octeţi	-32768 ÷ +32767
unsigned int	2 octeţi	0 ÷ +65535
signed long	4 octeţi	-2147483648 ÷ +2147483647
unsigned long	4 octeţi	0 ÷ +4294967295
float	4 octeţi	±1.176E-38 ÷ ±3.40E+38
double	8 octeţi	±1.7E-308 ÷ ±1.7E+308
pointer	1÷4 octeţi	adresa obiectului în funcţie de zona de memorie
Tipuri de date pentru SFR		
sbit	1 bit	0 sau 1
sfr	2 octeţi	0 ÷ +65535

Spaţiul de memorie

Compilatoarele dezvoltate suportă arhitectura completă a familiilor de microcontrolere pentru care sunt destinate. Fiecare variabilă poate fi asignată explicit unui anumit spaţiu de memorie.

Definirea spaţiului de memorie utilizat de variabile este prezentată în tabelul 1.2.a) pentru C51, respectiv 1.2.b) pentru C166.

Tabelul 2.2

Tipuri de memorie	Descriere
a) C51	
`code`	memorie program (64 kB); acces cu `MOVC @A+DPTR`
`data`	memorie internă adresabilă direct; acces foarte rapid la variabile (128 octeţi)
`idata`	memorie internă adresabilă indirect; posibilitate de adresare a întregului spaţiu intern de adrese (256 octeţi)
`bdata`	memorie internă adresabilă direct la nivel de bit; permite accesul la nivel de bit sau octet (16 octeţi)
`xdata`	memorie de date externă (64 kB); accesată folosind `MOVX @DPTR`
`pdata`	memorie de date externă adresată paginat (256 octeţi) folosind `MOVX @Rn`
b) C166	
`near`	pointer 16 biţi; calculul adresei pe 16 biţi permite accesul la: o 16 kB variabile în grupul `NDATA`; o 16 kB constante în grupul `NCONST`; o 16 kB date sistem în grupul `SDATA`.
`idata`	memorie internă adresabilă indirect; posibilitate de adresare a întregului spaţiu intern de adrese (2 kB)
`bdata`	memorie internă adresabilă direct la nivel de bit; permite accesul la nivel de bit sau octet (256 octeţi)
`sdata`	spaţiul sistem (0C000h...0FFFFh); permite definirea obiectelor PEC
`far`	pointer 32 biţi, permiţând accesul complet la întregul spaţiu de

	memorie; dimensiunea unui şir sau structuri este limitată la 16 kB
huge	pointer 32 biţi, permiţând accesul complet la întregul spaţiu de memorie; dimensiunea unui şir sau structuri nu mai este limitată la 16 kB

Modelul de memorie

Modelele de memorie determină modul în care sunt folosite variabilele dinamice, transmiterea parametrilor funcţiilor şi alte declaraţii fără un mod explicit de model de memorie. Parametrii şi variabilele automate sunt plasaţi în registrele procesorului disponibili apoi în memorie la adrese fixe în funcţie de modelul de memorie folosit.

C51 suportă trei modele de memorie iar C166 suportă cinci modele de memorie. Cu excepţia modelului TINY, toate celelalte lucrează în mod segmentat.

Caracteristicile modelului de memorie pentru C51 şi C166 sunt prezentate în tabelul 3.3.a), respectiv 3.3.b).

Tabelul 2.3

Model memorie	Declaraţii variabile	Declaraţii funcţii
a) C51		
SMALL	Parametrii şi variabilele automate sunt plasate în memoria internă dacă nu mai sunt registre disponibile (maxim 120 octeţi în memoria DATA)	
COMPACT	Parametrii şi variabilele automate sunt plasate în memoria externă dacă nu mai sunt registre disponibili. (maxim 256 octeţi în memoria PDATA)	
LARGE	Parametrii şi variabilele automate sunt plasate în memoria externă dacă nu mai sunt registre disponibili. (maxim 64 kB în memoria XDATA)	
b) C166		
TINY	16 biţi	16 biţi
SMALL	near (16 biţi)	near (16 biţi)
COMPACT	far (32 biţi)	near (16 biţi)
MEDIUM	near (16 biţi)	far (32 biţi)
LARGE	far (32 biţi)	far (32 biţi)

Pointeri

C51 suportă pointerii generici şi specifici tipului de memorie enumeraţi în tabelul 3.4.

Tabelul 2.4

Declarare	Memorie ocupată	Pointer la
float *p	3 octeţi	… float în tot spaţiul de memorie (pointer generic)
char data *p	1 octet	… char în meoria data
int idata *ip	1 octet	… int în memoria idata
long pdata *pp	1 octet	… long în memoria pdata
char xdata *xp	2 octeţi	… char în memoria xdata
int code *cp	2 octeţi	…. int în memoria code

La C166, spaţiile de memorie near, far şi huge pot fi aplicate şi pointerilor. Un pointer near permite adresarea oricărui obiect care foloseşte stiva sau este definit în zonele near, sdata, idata sau bdata. Un pointer far poate accesa toate obiectele din spaţiul de memorie al lui 80C16x,

dimensiunea obiectului fiind limitată la 16 kB. În situaţia pointerului huge, dimensiunea obiectului adresat poate fi mai mare de 16 kB.

Pentru modelele de memorie segmentate (toate, cu excepţia lui TINY), pentru adresarea pointerilor sunt folosite registrele DPPx, după cum urmează:

DPP0 pointeri huge şi far

DPP1 pointeri near grupul NCONST

DPP2 pointeri near grupul NDATA

DPP3 pointeri grupurile SDATA sau SYSTEM

Bancuri de registre şi mascarea registrelor

C51 suportă 4 bancuri de memorie utilizabile pentru programul curent. Definirea bancului de registre utilizat este făcută fie în linia de comandă, fie printr-o directivă #pragma, conform sintaxei:

REGISTERBANK (n)

Sintaxa pentru definirea bancului de registre folosit de o procedură C este:

[tip] nume_funcţie ([parametri]) [model] [reentrant] using n

unde n reprezintă numărul bancului de registre utilizat.

Mascarea registrelor permite o optimizare globală a utilizării registrelor interne. Folosirea acestei facilităţi îmbunătăţeşte programul prin eliminarea salvărilor inutile în stivă a registrelor unităţii centrale. Mascarea registrelor este activată de directiva REGFILE sau poate fi generată automat şi de editorul de legături L51.

Masca de registre este un cuvânt a cărei semnificaţie este:

	MSB															LSB
Registru	VAL	-	ACC	CY	PSW	B	DPL	DPH	R0	R1	R2	R3	R4	R5	R6	R7

VAL este setat pentru a indica o mască validă.

C166 suportă până la 128 de bancuri de registre logice, fiecare banc fiind compus din 16 registre R0...R15.

Adresa de bază a bancului de registre curent este dată registrul SFR CP (*Context Pointer*). Dacă o procedură este declarată cu atribut using, compilatorul generează codul necesar pentru schimbarea bancului de registre şi salvarea CP anterior, precum şi pentru revenirea la vechiul CP la terminarea rutinei.

Este recomandat ca această funcţie a compilatorului să fie utilizată în întreruperi sau pentru programele de aplicaţie în timp real.

Sintaxa pentru definirea bancului de registre este:

[tip] nume_funcţie ([parametri]) using *banc_registre*

unde *banc_registre* este un nume care identifică bancul de registre pe parcursul procesului de editare a legăturilor.

banc_registre poate fi folosit de funcţii diferite şi este chiar recomandabil ca, pentru întreruperile cu acelaşi nivel de prioritate, să fie folosit acelaşi banc de registre.

C166 suportă o mascarea a registrelor care poate fi aplicată prototipurilor sau declaraţiilor externe de funcţii. Mascarea registrelor specifică ce registre sunt modificate în timpul execuţiei funcţiei. Informaţia este utilizată de compilator pentru optimizarea alocării registrelor şi minimizarea timpului de execuţie şi a lungimii codului.

Un exemplu de utilizare al acestei facilităţi este prezentat în continuare:

`[extern][tip] nume funcţie ([parametri]) @=`*`reg_mask`*

unde: `[extern][tip] nume funcţie ([parametri])` este sintaxa standard de definire a funcţiei;

`reg_mask` este o valoare numerică, pe 16 biţi, corespunzătoare următoarei codificări:

	MSB															LSB
Registru utilizat	-	-	MDX	R12	R11	R10	R9	R8	R7	R6	R5	R4	R3	R2	R1	R0

Valoarea 0 a *`reg_mask`*, valoare implicită, specifică utilizarea tuturor registrelor. Valoarea 0x8000h indică faptul că nici un registru nu este afectat de funcţie. `MDX` reprezintă registrele `MDL`, `MDH` şi `MDC` utilizate de împărţire şi înmulţire.

Întreruperi

Compilatorul C51 are cuvinte specifice care permit programatorului un control complet asupra întreruperilor, bancurilor de registre utilizate şi a funcţiilor care vor fi apelate la sosirea unei cereri de întrerupere.

Specificarea adresei de start a vectorului de întrerupere pentru sistemele de dezvoltare la care programul de test se încarcă de la adresa 8000h trebuie folosită directiva `INTVECTOR/NOINTVECTOR` (fie în linia de comandă, fie cu `#pragma`).

Sintaxa unei funcţii de întrerupere este:

```
void nume_funcţie (void)[model][reentrant] interrupt n [using m]
```

unde `n` este numărul întreruperii şi `m` numărul bancului de registre utilizat.

Compilatorul C166 permite programatorului controlul complet asupra tuturor aspectelor referitoare la întreruperi sau bancurile de registre. C166 generează codul necesar pentru a efectua procedurile optime în întreruperi.

C166 oferă suportul necesar şi pentru canalele PEC. Zonele cu date PEC trebuie declarate în spaţiul de memorie `sdata`, sau trebuie declarate explicit în segmentul 0 de memorie. Pentru rezervarea zonelor de memorie necesare PEC trebuie folosită directiva `PECDEF` urmată de numărul canalelor utilizate (de exemplu: `#pragma PECDEF (1,3,4-7)`).

Sintaxa folosită pentru întreruperi este următoarea:

`void nume_funcţie (void) interrupt `*`definire_vector`*` [using `*`reg_mask`*`]`

unde: *`definire_vector`* reprezintă:

- identificator întrerupere = număr excepţie;
- identificator întrerupere;
- număr excepţie.

C166 permite definirea unui vector de întrerupere fie în formă simbolică, fie în formă absolută. Legătura între forma absolută şi simbolică este dată de relaţia:

adresă_vector = număr_excepţie * 4

Utilizarea procedurilor de întrerupere prezintă următoarele particularităţi:

- dacă nu este utilizată comutarea bancului de registre (atributul `using`), toate registrele utilizate în rutină sunt salvate în stivă;
- registrele SFR, `MDC`, `MDL` şi `MDH`, dacă sunt utilizate în rutină, sunt salvate automat în stivă iar la ieşire sunt restaurate;
- nu este posibilă trecerea sau întoarcerea de parametri;
- din întrerupere pot fi apelate orice funcţii standard C (toate sunt reentrante);
- funcţia este terminată de instrucţiunea `RETI`.

Transmiterea parametrilor

Cunoaşterea modului în care sunt transferaţi parametrii la apelul unei funcţii este utilă în cazul în care se scriu atât funcţii atât în asamblare cât şi în C.

C51 poate transfera până la trei parametrii folosind registrele procesorului, conform cu tabelul 3.5 Dacă nu sunt registre disponibile sau dacă se foloseşte directiva `#pragma NOREGPARAMS`, parametrii sunt transmişi folosind locaţii din memorie în funcţie de modelul de memorie folosit.

Tabelul 2.5

Tip parametru	char, pointer 1 octet	int, pointer 2 octeţi	long, float	pointer generic
Parametrul 1	R7	R6 şi R7	R4, R5, R6, R7	R1, R2, R3
Parametrul 2	R5	R4 şi R5	R4, R5, R6, R7	R1, R2, R3
Parametrul 3	R3	R2 şi R3	–	R1, R2, R3

Valoarea întoarsă de funcţie se află în registrele unităţii centrale, după cum se prezintă în tabelul 3.6

Tabelul 2.6

Tipul valorii returnate	Registrul
Bit	carry
(unsigned) char	R7
(unsigned) int	R6 (MSB), R7 (LSB)
(unsigned) long	R4 (MSB), R5, R6, R7 (LSB)
Float	R4, R5, R6, R7 (format IEEE pe 32 de biţi)
pointer generic	R1, R2, R3 (R3 selector tip memorie, MSB în R2, LSB în R1)

C166 permite transferul a maxim cinci parametri prin intermediul registrelor de uz general ale unităţii centrale `R8...R12`. Dacă toate cele cinci registre sunt utilizate, va fi folosită stiva sistem. De asemenea, stiva este folosită pentru păstrarea variabilelor automate şi poate fi accesată prin intermediul registrului `R0`. Pentru transmiterea valorilor de tip bit este folosit `R15`.

Valorile returnate de proceduri sunt păstrate în registre fixe (prezentate în tabelul 3.7, în funcţie de natura acestora. În acest mod, interfaţarea cu programele scrise în asamblor este foarte mult uşurată.

Tabelul 2.7

Valoare returnată	Registru	Observaţii
`Bit`	`R4.0`	
`(unsigned) char`	`RL4`	
`(unsigned) int`	`R4`	
`(unsigned) long`	`R4,R5`	LSB în `R4`, MSB în `R5`
`Float`	`R4,R5`	format 32 biţi IEEE. exponent şi semn în `R5`
`Double`	`R4...R7`	format 64 biţi IEEE. exponent şi semn în `R7`
`near pointer`	`R4`	
`far/huge pointer`	`R4,R5`	deplasament în `R4`, selector în `R5`

Fişiere de configurare

Un proiect scris în C51 trebuie legat (linkeditat) cu fişierul STARTUP.a51 care conţine o rutină de iniţializare a zonelor de memorie folosite de rutinele C51 şi un salt la funcţia `main()`.

Principala modificare adusă fişierului de configurare STARTUP.a51 pentru C51 este înlocuirea declaraţiei: `CSEG AT 0` cu `CSEG AT 8000h` atunci când programele sunt executate din memoria RAM (de la adresă 8000h) şi programul monitor este de la adresă 0h.

Compilatorul C166 poate fi adaptat la diferite sisteme de dezvoltare prin intermediul a câteva fişiere. Fişierele de configurare şi semnificaţia lor este:

`STARTUP.A66` conţine câteva definiţii utilizate la iniţializarea sistemului:

`_MCTC` setează registrul `BUSCON.0...BUSCON.3`; defineşte numărul de cicluri de aşteptare la accesul memoriei;

`_RWDC` setează registrul `BUSCON.4`; defineşte întârzierea semnalelor $\overline{RD}$/$\overline{WR}$;

`_MTTC` setează registrul `BUSCON.5`; programează intervalul de înaltă impedanţă;

`_RDYEN` setează registrul `BUSCON.12`; validează utilizarea semnalului $\overline{READY}$ pentru terminarea ciclului de memorie;

`_CLKEN` setează registrul `SYSCON.8`; validează producerea semnalului `CLKOUT` pe pinul `P3.15`;

`_BYTDIS` setează registrul `SYSCON.9`; validează producerea semnalului $\overline{BHE}$ pe pinul `P3.12`;

`_SGTDIS` setează registrul `SYSCON.11`; invalidează segmentarea (pentru modelul de memorie `TINY`);

`_STKSZ` setează registrul `SYSCON.13...SYSCON.15`; defineşte mărimea stivei între 32 şi 1024 cuvinte;

`EXT_RAM` validează semnalul $\overline{WR}$ pe pinul `P3.13`;

`WATCHDOG` dezactivează timerul de iniţializare;

`CLR_MEMORY` validează ştergerea memoriei RAM la iniţializare;

`INIT_VARS` validează iniţializarea variabilelor declarate explicit.

Mai multe informaţii despre registrele de configurare pot fi găsite în paragraful 2.3.1, *Caracteristici programabile ale magistralei* şi 2.2.3, *Registrele speciale ale unităţii centrale*.

`PUTCHAR.C` conţine câteva rutine utilizate pentru transmiterea de caractere. Fişierul respectiv foloseşte interfaţa ASC a microcontrolerului pentru comunicare cu un terminal cu protocol XON/XOFF.

`GETKEY.C` conţine câteva rutine utilizate pentru recepţionarea de caractere. Fişierul respectiv foloseşte interfaţa ASC a microcontrolerului pentru comunicare cu un terminal. Nu realizează funcţii de conversie.

Toate funcţiile de intrare ieşire, ca de exemplu `printf`, `puts`, `scanf`, folosesc funcţiile `putchar` sau `getkey`. Prin modificarea acestor fişiere şi introducerea lor în proiectul dezvoltat se poate face adaptarea funcţiilor de intrare/ieşire la echipamentele periferice folosite. De asemenea, se pot implementa diverse protocoale de comunicaţie.

2.1.2 Asamblorul

Asamblorul conţine programele şi instrumentele necesare pentru programarea familiilor de procesoare pentru care sunt destinate: A51 pentru 8xC552 şi A166 pentru 80C16x.

Rolul asamblorului este de a transforma fişierul sursă (de regulă .asm) în fişier relocatabil (.obj).

Invocarea asamblorului poate fi destul de complexă, întrucât pot fi specificate multe opţiuni. Informaţii suplimentare se pot găsi în lucrările *A51 ASSEMBLER User's Guide, Keil Elektronik GmbH* şi *A166 ASSEMBLER User's Guide, Keil Elektronik GmbH*.

Sintaxa invocării este:

A51 *fişier.asm* [*opţiuni*]

A166 *fişier.asm* [*opţiuni*]

Detalii suplimentare despre opţiunile specifice asambloarelor A51 şi A166 sunt prezentate în paragraful *Controlul asamblorului.*

Operanzi şi expresii

Programul sursă pentru asamblor constă în linii de mnemonice având o formă asemănătoare cu linia următoare:

```
[etichetă]:      mnemonic   [expr1] [,expr2] [,expr3]   [;comentariu]
```

Semnificaţia elementelor de mai sus este următoarea:

`etichetă:` o valoare simbolică a adresei fizice a instrucţiunii utilizată pentru salturi, apeluri de subrutine şi depanatoare;

`mnemonic` un set de caractere recunoscut de asamblor ca instrucţiune a unităţii centrale;

`expr1…3` operanzii asociaţi instrucţiunii respective.

Operanzii acceptaţi de A51 şi A166, care pot fi în număr de la 0 până la 3, sunt prezentaţi în tabelul 3.8.a), respectiv 3.8.b).

Tabelul 2.8

Operand	Semnificaţie
a) A51	
A	acumulator
R0…R7	registre de uz general din bancul curent de registre
DPTR	pointer la memoria de date internă sau externă
PC	contor program (conţine adresa următoarei instrucţiuni)
C	indicator deplasare (carry)
AB	registru dublu A+B folosit de instrucţiunile MUL şi DIV
AR0…AR7	registre folosite pentru adresare absolută
#DATAn	constantă imediată pe 8 sau 16 biţi
b) A166	
Rn,Rm	acces direct la GPR din bancul curent de registre R0…R15
REG	acces direct la orice registru GPR sau SFR
BITWORD	acces la cuvânt în spaţiul de memorie adresabil la nivel de bit
BITADDR	acces la bit în spaţiul de memorie adresabil la nivel de bit
MEM	acces la orice locaţie de memorie
[Rn],[Rm]	acces indirect la întreaga memorie funcţie de conţinutul GPR
CADDR	adresă pe 16 biţi a codului instrucţiunii într-un segment de 64 kB
REL	deplasament pentru salturi relative. Valoarea este de +127/-128 cuvinte relativ la deplasamentul curent
SEG	numărul segmentului de cod
#TRAP	constantă pe 7 biţi folosită ca număr al vectorului de întrerupere
CC_cond	cod condiţie pentru salturi, apeluri rutine etc.

Operanzii instrucţiunilor sunt expresii primare. Expresiile constau în numere, simboluri şi operatori.

În continuare, vor fi prezentate numai particularităţile specifice asambloarelor A51 şi A166:

- Numerele pot fi reprezentate în cod hexazecimal (h,H), zecimal (d,D), octal (o,O,q,Q) sau binar (b,B). Sunt admise numai şiruri de unul sau două caractere ASCII.
- Simbolurile reprezintă valori numerice sau adrese. Se pot utiliza ca valori numerice în expresii unde sunt folosite constante numerice.

  Pot fi definite de utilizator sau pot fi speciale, cuvinte rezervate (GPR, "ORG", "$" etc.).

  Simbolurile pot avea la C166 următoarele atribute: TYPE (tipul simbolului – bit, byte, word, data3 etc.; folosite pentru determinarea tipului accesului şi selectarea instrucţiunii potrivite – de exemplu MOVB sau MOVW), SECTION (segmentul de memorie care conţine cod, constante sau variabile), SCOPE (întinderea simbolului – local pentru module, global, public sau extern pentru constante şi simboluri relocatabile), VALUE (valoarea numerică sau deplasamentul simbolului) şi CHANGEABLE (simbolurile definite cu directiva SET pot fi modificate pe parcursul programului).

Simbolurile pot fi relocatabile, adică ele sunt evaluate în momentul editării legăturilor. Aceste simboluri sunt: numele de secţiuni, numele de grupuri, variabilele şi etichetele, constantele externe.

Operatorii sunt prezentaţi în tabelul 3.9. (cu ¤ sunt marcaţi operatorii admişi numai de C166).

Tabelul 2.9

Operator	Semnificaţie
`()`	schimbarea ordinii de evaluare
`.`	separator poziţie bit
`BIT PTR,BYTE PTR, WORD PTR,BITWORD PTR,NEAR PTR,FAR PTR`	¤ tip pointer
`DATA3,DATA4,DATA8 DATA16`	constantă 3 biţi (0-7), 4 biţi (0-15), 8 biţi (0-256), respectiv 16 biţi (0-65535)
`DPP0-DPP3`	¤ indicator pagină de date
`SEG`	¤ număr segment variabilă (64 kB)
`PAG`	¤ număr pagină variabilă (16 kB)
`SOF`	¤ deplasament segment variabilă
`POF`	¤ deplasament pagină variabilă
`BOF`	¤ poziţie bit
`HIGH,LOW`	valoarea octetului superior, respectiv inferior
`NOT`	complement
`+,-`	semnul expresiei
`*,/,MOD`	înmulţire, împărţire, rest modulo
`+,-`	adunare, scădere
`SHL(<<),SHR(>>)`	deplasare stânga, dreapta
`AND(&),OR(|), XOR (^)`	ŞI LOGIC, SAU LOGIC, respectiv SAU EXCLUSIV LOGIC
`LT(<),LE(<=) GT(>),GE(>=)`	comparare: mai mic, mai mic sau egal, mai mare, respectiv mai mare sau egal
`ULT, ULE, UGT, UGE`	¤ comparări fără semn: mai mic, mai mic sau egal, mai mare, respectiv mai mare sau egal
`SHORT`	¤ generarea unui salt relativ sau apel la subrutină în domeniul –128/+127 cuvinte

Directive

Directivele asamblorului sunt folosite pentru controlul procesului de asamblare înainte de transformarea fişierului sursă în cod maşină.

Directivele asamblorului A51 se pot clasifica după cum urmează:

- definiţii de simboluri:

`nume SEGMENT tip [relocatabil]`

  - `nume` – numele segmentului definit;
  - `tip` – indică spaţiul de memorie utilizat; poate fi `CODE` (memorie program), `XDATA` (memorie externă), `DATA` (memorie date internă), `IDATA` (memorie internă adresabilă indirect) şi `BIT` (memorie adresabilă la nivel de bit);
  - `relocatabil` – poate fi `PAGE` (segmentul începe la adrese divizibile cu 256), `INPAGE` (segmentul trebuie inclus într-un bloc de 256 octeţi), `INBLOCK` (segmentul trebuie inclus într-un bloc de 2048 octeţi), `BITADDRESABLE`, (segmentul trebuie inclus în zona adresabilă la nivel de bit), `UNIT` (este adresa iniţială, de tip bit sau octet, pentru un segment bit, respectiv toate celelalte tipuri), `OVERLAYABLE` (segmentul poate fi suprapus cu alte segmente);

`simbol xxxx expresie`

  - `simbol` – numele segmentului definit;

- xxxx – poate fi CODE, XDATA, DATA, IDATA şi BIT şi semnifică tipul spaţiului de memorie alocat simbolului;
- expresie – constă într-o valoare numerică relocatabilă; nu trebuie să conţină referinţe ulterioare;

Dacă xxxx este EQU sau SET, atunci simbolul sau registrul intern primeşte valoarea expresiei (care este fixă – în cazul EQU sau poate fi modificată ulterior – în cazul SET).

- iniţializarea şi rezervarea spaţiului de memorie:

[etichetă:] xx expresie_numerică

- etichetă – reprezintă adresa simbolică începând cu care este rezervat spaţiul de memorie;
- xx – poate fi DS (rezervă un spaţiu egal cu expresie_numerică în memoria internă sau externă), DBIT (rezervă un număr de biţi egal cu expresie_numerică în memoria internă adresabilă la nivel de bit), DS (iniţializează un octet din memoria program cu expresi-e_numerică), DW (iniţializează doi octeţi din memoria program cu expresie_numerică).

- editarea legăturilor programului:

PUBLIC simbol [,simbol [,…]]

PUBLIC declară numele unui simbol care în alte module este definit EXTRN.

EXTRN tip_segment (simbol),…

EXTRN declară numele simbolurilor utilizate în alte module. Fiecare simbol extern are definit un tip de segment (CODE, DATA, IDATA XDATA, BIT sau NUMBER – fără tip) care defineşte utilizarea simbolului.

- controlul stării asamblorului şi selectarea segmentului:

END – este ultima linie dintr-un program sursă, marcând sfârşitul acestuia;

ORG expresie – este folosită pentru modificarea contorului de adrese la valoarea expresie pentru a stabili adresa de început a modulului;

RSEG segment – selectează un segment definit anterior şi îl foloseşte ca segment de lucru;

xSEG [AT adresă]– xSEG poate fi CSEG,DSEG,XSEG,ISEG BSEG; este folosit pentru declararea segmentelor absolute de la adresă; dacă AT adresă nu este specificat, este continuat ultimul segment; dacă nu este selectat nici un segment, este creat un nou segment începând cu adresa 0h;

USING expresie – notifică asamblorului care banc de registre va fi utilizat.

Directivele asamblorului A166 se pot caracteriza ca:

- definiţii de secţiuni (SECTION):

nume SECTION tip_secţiune[tip_aliniere][tip_combinare][nume_clasă]

- nume – numele secţiunii;
- tip_secţiune – poate fi CODE, DATA sau BIT;
- tip_aliniere – secţiunea poate fi nealiniată (implicit) sau aliniată la nivel de BIT, BYTE, WORD (la adrese pare), DWORD (la adrese divizibile cu 4), PAGE (aliniată la 16 kB), SEGMENT (aliniată la 64 kB), BITADDRESSABLE (la adrese pare în zona 0FD00h-0FDFEh) sau PECADDRESSABLE (la adrese pare în zona 0FDE0h-0FDFEh);
- tip_combinare – specifică dacă secţiunea este sau nu combinată cu secţiuni din alte module; A166 admite următorii specificatori: PRIVATE (necombinat), PUBLIC (secţiunile care au acelaşi nume vor fi combinate într-o singură secţiune), GLOBAL (determină vizibilitatea secţiunii sau simbolurilor pentru întreaga aplicaţie), COMMON (toate secţiunile cu acelaşi nume şi acest atribut vor fi suprapuse în memorie formând o singură secţiune), SYSSTACK (toate secţiunile cu acelaşi nume şi acest atribut vor fi combinate într-o singură secţiune care constituie stiva sistem), USRSTACK (asemănător cu SYSSTACK, numai că stiva este dispusă oriunde în

memorie şi este adresată cu DPPn:deplasament), GLBUSRSTAK (similar cu USRSTACK), AT (folosit pentru plasarea secţiunii la o adresă absolută).

- definiţii de grupuri (GROUP):

nume CGROUP/DGROUP nume_secţiune [,nume_secţiune [,…]]

- nume – numele grupului;
- CGROUP/DGROUP – specifică dacă este grup de cod, respectiv de date;
- nume_secţiune – secţiunile care aparţin membrilor grupului; există posibilitatea alocării unei pseudo-secţiuni SYSTEM care specifică o secţiune absolută în pagina 3.

- setare DPPn (ASSUME):

ASSUME DPPn: [obiect]

- DPPn – poate fi DPP0, DPP1, DPP2 sau DPP3;
- obiect – reprezintă domeniile care vor fi adresate cu unul din registrele DPPn, şi anume: nume_secţiune (inclusiv SYSTEM), nume_grup, nume_variabilă sau _simbol.

- rezervare resurse (REGDEF, REGBANK, SSKDEF, PECDEF, PROC/ENDP, DEFR/DEFA/DEFB):

nume_banc_registre REGBANK [domeniu] [,…]

- nume_banc_registre – numele bancului de registre;
- domeniu – registrele care vor fi comutate de instrucţiunea SCXT.

[nume_banc_registre] REGDEF domeniu [,…]

- aceeaşi semnificaţie cu REGBANK.

SSKDEF mărime_stivă

- mărime_stivă – poate avea valorile 0, 1, 2 sau 3, stiva având mărimea de 256, 128, 64 respectiv 32 cuvinte în zonele 0FA00h-0FBFFh, 0FB00h-0FBFFh, 0FB80h-0FBFFh, respectiv 0BFC0h-0FBFFh.

PECDEF nr_canal [,nr_canal [,…]]

- nr_canal – reprezintă numărul canalului PEC pentru care este rezervată memoria.

PROC/ENDP sunt folosite împreună pentru a defini o etichetă pentru o secvenţă de instrucţiuni denumite procedură.

```
nume PROC [tip]
…
nume ENDP sau
nume PROC TASK [nume_task][INTNO [nume_intr][=nr.intr]]
```

- nume – reprezintă numele procedurii;
- tip – poate fi NEAR (implicit) sau FAR;
- TASK – defineşte un nume de task nume_task (un modul funcţional de program care este activat de software sau excepţie hardware care produce o întrerupere nume_intr cu vectorul nr.intr).

DEFR/DEFA/DEFB sunt folosite pentru definirea unor nume personalizate de SFR, adrese absolute interne din RAM, respectiv locaţii adresabile la nivel de bit.

- definiţii de simboluri (EQU, SET, BIT, LIT):

nume EQU expresie

- asigură pentru simbolul nume valoarea rezultată din expresie.

nume SET expresie

- creează un nou simbol nume care poate fi redefinit cu valoarea rezultată din expresie.

nume BIT expresie

- asignează valoarea expresie variabilei de tip bit nume.

nume LIT 'şir_caractere'

– asigură substituirea identificatorului nume cu textul 'şir_caractere'.

- rezervare memorie (DBIT, DS, DSB, DSW):

  [nume[:]] Dxxx expresie – alocă variabilei nume un număr de biţi (DBIT) sau octeţi (DS) egali cu valoarea expresie. DSB şi DSW alocă numărul de octeţi sau cuvinte corespunzător dar nu se pot utiliza în secţiuni de tip bit.
- iniţializare memorie (DB/DW, DBPTR/DSPTR/DPPTR):

  [nume[:]] Dx valoare – iniţializează un octet (DB) sau un cuvânt (DW) cu mărimea valoare.

  [nume[:]] DxPTR valoare – iniţializează un pointer de segment (DSPTR), de pagină (DPPTR) sau de bit (DBPTR), cu mărimea valoare.
- legături ale programului (PUBLIC/GLOBAL, EXTERN):

  PUBLIC nume [,nume[,…]]

  GLOBAL nume [,nume[,…]]

  EXTERN [DPPn] nume:tip [,…] – specifică simbolurile care vor fi referite în modulul curent dar sunt definite în alt modul (în care simbolurile sunt definite ca PUBLIC sau GLOBAL)
- controlul asamblării (ORG, EVEN):

  ORG expresie – modifică contorul de program în interiorul secţiunii curente. Valoarea expresie setează contorul la valoarea dorită relativ faţă de adresa de start a secţiunii.

  EVEN – asigură alinierea codului sau datelor următoare la adrese pare.

Controlul asamblorului

Elementele de control ale asamblorului sunt destinate să altereze comportarea normală a acestuia. Pot fi specificate fie în linia de comandă, fie în programul sursă prin intermediul directivelor '$'.

Cele mai importante comenzi utilizate de A51 sunt prezentate în continuare:

- (NO)MOD51 specifică faptul că toate numele descrise în specificaţia hard-ware pentru 8xC51 sunt cunoscute asamblorului. Dacă se lucrează cu alt controler trebuie folosită comanda NOMOD51 urmată de o comandă INCLUDE pentru redefinirea registrelor speciale.
- (NO)DEBUG informează asamblorul să introducă în fişierul de ieşire informaţiile despre simboluri pentru a fi utilizate de programele de depanare.
- (NO)REGISTERBANK specifică bancurile de registre utilizate de modul.
- INCLUDE (fişier) inserează conţinutul fişier în programul sursă. fişier poate conţine definiţii, programe sursă etc.

Cele mai importante comenzi utilizate de A166 sunt prezentate în continuare:

- (NO)DEBUG informează asamblorul să introducă în fişierul de ieşire informaţiile despre simboluri pentru a fi utilizate de programele de depanare.
- INCLUDE inserează conţinutul fişierului specificat în programul sursă, imediat după linia de control,

- (NO)MOD166 specifică faptul că toate numele descrise în specificaţia hardware pentru 80C166 sunt cunoscute asamblorului. Dacă se lucrează cu alt controler trebuie folosită comanda NOMOD166 urmată de o comandă INCLUDE pentru redefinirea registrelor speciale.
- (NO)SEGMENTED specifică asamblorului dacă va lucra în mod nesegmentat (memorie adresabilă de maxim 64 kB) sau segmentat.
- (NO)SYMBOLS specifică asamblorului să introducă în fişierul .lst lista de simboluri.
- (NO)XREF indică asamblorului să creeze un tabel de referinţe încrucişate care indică utilizarea simbolurilor, tabel care este adăugat listei de simboluri. Tabelul de referinţe încrucişate constă într-o listă cu numerele de linie în care sunt întâlnite simbolurile utilizate în program.

2.1.3 Editorul de legături

Editorul de legături combină mai multe module obiect (.obj) diferite într-un singur modul absolut (.m66) care poate fi încărcat de un depanator sau emulator. El rezolvă referinţele publice şi externe şi părţile de program relocatabile sunt asignate la adrese absolute. De asemenea, editorul alege librăriile necesare şi leagă codurile rutinelor din librării la programul final. În final, editorul de legături produce un fişier obiect absolut care conţine întregul program, eventual şi informaţiile necesare pentru depanare. Informaţii suplimentare despre editoarele de legături pot fi găsite în lucrările *8051 Utilities User's Guide, Keil Elektronik GmbH* şi *80C166 Utilities User's Guide, Keil Elektronik GmbH*.

Pentru familiile 8xC552 şi 80C16x se folosesc editoarele L51, respectiv L166.

Sintaxa comenzilor este următoarea:

L51 *listă_intrare* [TO *fişier_ieşire*] [*listă_control*]

L166 *listă_intrare* [TO *fişier_ieşire*] [*listă_control*]

unde: *listă_intrare* reprezintă o listă de fişiere separate prin virgule. Fişierele conţin modulele de programe relocatabile care vor fi combinate pentru a rezulta modulul de program absolut.

fişier_ieşire este numele sub care va fi scris modulul de program absolut. Dacă nu este introdus nici un nume, primul nume din *listă_intrare* va fi folosit pentru programul final.

listă_control conţine comenzile şi parametrii din linia de apel.

Cele mai importante comenzi din *listă_control* utilizate de L51 sunt prezentate în continuare:

- ixref produce un fişier cu lista referinţelor încrucişate;
- ramsize (valoare) specifică mărimea memoriei RAM interne în octeţi;
- precede,bit,data,idata,stack,xdata,code permit definirea adreselor diferitelor spaţiilor de memorie, conform cu tabelul 1.10:

Tabelul 2.10

Parametru	Spaţiu adrese	Domeniu adrese	Tip segment
precede	bancuri registre şi memorie adresabilă la nivel de bit	00h-2Fh	DATA, IDATA
bit	memorie adresabilă la nivel de bit	00h-7Fh	BIT, DATA, IDATA
data	memorie adresabilă direct	00h-7Fh	DATA, IDATA
idata	memorie adresabilă indirect	00h-FFh	IDATA
stack	memorie adresabilă indirect	00h-FFh	IDATA
xdata	memorie externă	0000h-FFFFh	XDATA
code	memorie program	0000h-FFFFh	CODE

- pdata (valoare) specifică folosirea pentru adresarea memoriei XDATA a portului P2. Pentru a folosi această metodă de adresare utilizatorul trebuie să modifice corespunzător fişierul de configurare STARTUP.A51 astfel încât adresa paginată să fie setată corect.

Cele mai importante comenzi din *listă_control* utilizate de L166 sunt prezentate în continuare:

- CLASSES specifică un domeniu de adrese fizice sau ordinea de alocare pentru toate secţiunile cu un nume de clasă dat. CLASSES permite definirea simplă a structurii memoriei microcontrolerului. Valoarea implicită este:

```
CLASSES  (NCONST(0,03FFF), NCODE(0,F9FF), NDATA(04000,07FFF),
          NDATA0(04000,07FFF), SDATA(0C000,0FFFF), SDATA0(0C000,0FFFF),
          IDATA(0FA00,0FDFF), IDATA0(0FA00,0FDFF), BIT(0FD00,0FDFF),
          BIT0(0FD00,0FDFF), BDATA(0FD00h,0FDFF), BDATA0(0FD00,0FDFF))
```

- GROUPS specifică o adresă de start sau ordinea de alocare pentru un grup. Toate secţiunile care nu sunt declarate cu directiva SECTIONS şi care sunt membre ale grupului respectiv sunt alocate conform argumentelor din directiva GROUPS.

  Exemplu: L166 prog.obj GROUPS (NDATA (0x1000), NCONST)

  Toate secţiunile membre ale grupului NDATA sunt alocate la adresa 0x1000, după care este alocat şi grupul NCONST.
- NODEFAULTLIBRARY dezactivează legarea automată a bibliotecilor C la programul final. În situaţia în care este folosită această directivă, programatorul trebuie să adauge manual în linia de comandă numele bibliotecilor care sunt necesare pentru linkeditare.
- NOINIT dezactivează ştergerea memoriei RAM interne la iniţializare.
- NOMAP elimină harta memoriei şi lista grupurilor din fişierul listă de ieşire. Harta memoriei conţine informaţii despre structura memoriei fizice şi afişează asignarea adreselor pentru secţiunile programului.
- PURGE elimină complet informaţiile de depanare din fişierul listă de ieşire.
- RESERVE anunţă L166 să evite folosirea adreselor în zona de memorie specificată, de la *adresă_start* până la *adresă_sfârşit*.

  Sintaxă: RESERVE (adresă_start - adresă_sfârşit)
- SECTIONS defineşte o adresă fizică de memorie sau ordinea secţiunilor specificate. Toate secţiunile definite în această directivă sunt alocate

secvenţial. Prima secţiune este alocată la cea mai mică adresă (de regulă 0) sau la valorile implicite declarate prin directivele CLASSES sau GROUPS.

Sintaxă: SECTIONS(nume_secţiune[%nume_clasă](adresă))

Exemplu: L166 prog.obj SECTIONS(?PR?PROG%NCODE(0x1200),&?CO?PROG%NCONST)

Secţiunea ?PR?PROG şi clasa NCODE sunt alocate la adresa 0x1200, după care sunt alocate secţiunea ?CO?PROG şi clasa NCONST la adrese superioare valorii 0x1200.

2.1.4 Programe utilitare

Cu excepţia compilatoarelor, asambloarelor şi editoarelor de legături, există numeroase alte unelte software care uşurează munca programatorilor. Vor fi prezentate în continuare numai administratorul de biblioteci (LIB51 şi LIB166) şi convertorul de fişiere obiect în fişiere Intel hex (OHS51 şi OH166).

Administratorul de biblioteci

Administratorul de biblioteci permite crearea şi modificarea bibliotecilor utilizate de editorul de legături. Modul de utilizare a celor două programe LIB51 şi LIB166 este identic, utilizatorul trebuind să modifice în linia de comandă numai numele programului apelat.

Sintaxa liniei de comandă este:

```
LIBxxx [comandă]
```

unde:

LIBxxx reprezintă LIB51 sau LIB166, funcţie de familia de microcontrolere utilizată;

comandă poate lipsi, caz în care este afişat promptul '*' pentru a indica aşteptarea unei comenzi, sau este una din următoarele instrucţiuni:

- ADD nume_fişier [(nume_modul,…)][,…] TO bibliotecă – adaugă modulul (modulele) nume_modul din fişierul nume_fişier la librăria bibliotecă;
- CREATE bibliotecă – creează o nouă bibliotecă vidă;
- DELETE bibliotecă (nume_modul,…) – şterge din bibliotecă modulul (modulele) nume_modul,….
- LIST bibliotecă [TO fişier_listă][PUBLICS] – afişează la consolă sau în fişier_listă conţinutul bibliotecă, însoţite eventual de o listă a modulelor cu atributul PUBLICS.

Convertorul fişiere obiect-hexazecimal

Convertorul de fişiere obiect în fişiere Intel hex (OHS51 şi OH166) permite transformarea fişierelor absolute (.obj) în fişiere Intel hex transferabile pe interfaţa serială către memoria RAM a sistemului de dezvoltare.

Sintaxa liniei de comandă este asemănătoare pentru ambele familii de microcontrolere:

```
OHxxx fişier_obiect
```

unde:

Ohxxx reprezintă OH51 sau OH166;

fişier_obiect este numele fişierului .obj care va fi convertit în fişier hex.

OH166 poate avea şi argumentul [H167] care indică programului folosirea unui format Intel hex 386, format necesar pentru sistemele cu 80C167 care au memorii EPROM la adrese peste 0F'FFFh.

2.1.5 Depanatoare

Firma Franklin a realizat pachete de programe de depanare dezvoltate pe calculatoare compatibile PC, pentru ambele familii de microcontrolere: dScope-51 şi dScope-166.

dScope este un depanator simbolic pentru limbaje de nivel înalt cu o interfaţă utilizator orientată pe ferestre DOS. Utilizarea este uşurată prin folosirea de meniuri derulante sau linii de comandă.

Ambele pachete au o structură asemănătoare, fiecare conţinând un simulator hardware (DS51, respectiv DS166) şi o interfaţă cu monitorul sistemului de dezvoltare (TS51, respectiv TS166).

Programele dezvoltate pot fi testate integral, toate funcţiile perifericelor (timere, convertoare A/D etc.) fiind simulate. Depanatorul permite încărcarea fişierelor în format Intel hex existând posibilitatea executării lui pas cu pas, introducerea de breakpoint-uri, vizualizarea conţinutului SFR, memoriei şi a unor variabile, structuri sau şiruri de date.

Pachetele de programe dScope-51 şi dScope-166 sunt extrem de vaste şi prezentarea lor *in extenso* depăşeşte scopul acestei lucrări. Utilizatorilor li se recomandă consultarea referinţelor bibliografice *dScope-51 User's Guide, Keil Elektronik GmbH* şi *DScope 166 User's Guide, Keil Elektronik GmbH*.

2.1.6 Monitoare

Monitoarele folosite pentru dezvoltarea aplicaţiilor cu microcontrolere sunt formate din două componente distincte: un monitor tip terminal şi un monitor PROM.

Monitorul de tip terminal este un program DOS lansat pe calculatorul PC care controlează sistemul de dezvoltare. Principalele facilităţi asigurate de monitoarele MON51 (pentru familia 8xC552) şi MON166 (pentru familia 80C16x) sunt:

- selectarea şi parametrii portului serial de interfaţare (definiţi în linia de comandă):

  `MONxx [parametru]` – parametru reprezintă `COM1…COM4` (numele portului serial utilizat), `INT14` (comunicaţia serială se face folosind întreruperea BIOS 14h), `NOINT` (comunicaţia este făcută fără întreruperi hardware) sau `baudrate (valaore)` (valorile admise sunt 300, 600, 1200, 2400, 4800, 9600 şi 19200 biţi/s).
- încărcarea unor fişiere Intel hex în memoria sistemului de dezvoltare:

  Încărcarea unui fişier este făcută conversaţional, la comanda <F2> sau <Alt>2, monitorul răspunzând cu: (`Input File: …`).
- administrarea unor puncte de întrerupere (*break point*):

  `BS adresă` – definire adresă break point;

  `BK număr` – ştergere globală break point (cu parametrul ALL) sau numai anumite puncte funcţie de număr;

BL – afişare asociere între adresă şi număr break point;

BE număr – validare globală break point (cu parametrul ALL) sau numai anumite puncte funcţie de număr;

BD număr – invalidare globală break point (cu parametrul ALL) sau numai anumite puncte funcţie de număr.

- lansarea în execuţie a unui program din memoria RAM a sistemului de dezvoltare:

  g [adresă_start[,adresă_final]].

- execuţia pas cu pas a unui număr de instrucţiuni de program din memoria RAM a sistemului de dezvoltare:

  t [număr_paşi] – număr_paşi reprezintă câţi paşi vor fi executaţi la o comandă.

- este permisă afişarea, modificarea, asamblarea, dezasamblarea unor porţiuni din memoria internă, externă, date sau program.

De asemenea, monitorul terminal este considerat consolă de către programele aplicative care folosesc funcţiile PUTCHAR.C sau GETKEY.C.

Pentru interfaţarea între sistemul de dezvoltare şi PC-AT (conector DIN25) este recomandată o cablare ca în figura 3.

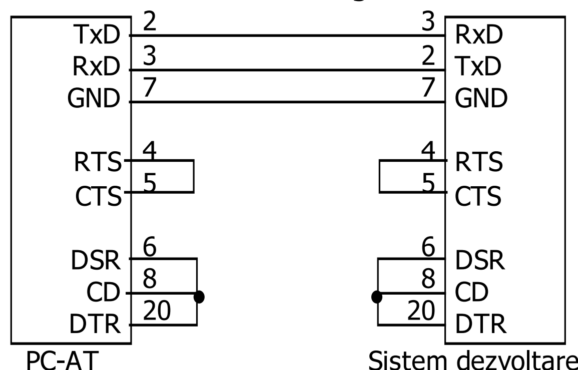


Figura 2.1. Interfaţarea PC cu sistemul de dezvoltare

Monitorul PROM este destinat să asigure rutinele esenţiale ale sistemului cu sistemul de dezvoltare, protocolul de legătură pe interfaţă RS232 cu calculatorul PC, utilizatorul având posibilitatea de a adăuga şi alte programe necesare.

Monitorul PROM poate fi personalizat de utilizator prin intermediul fişierelor de configurare.

Sistemele cu 80C16x, datorită facilităţii *Bootstrap Loader* (descrisă în paragraful 2.13), se pot dispensa, cel puţin în faza de dezvoltare a aplicaţiei, de memoriile PROM, programele monitor fiind încărcate în faza de iniţializare direct de pe PC.

2.2. Sisteme de dezvoltare

Sistemele de dezvoltare descrise în acest paragraf au fost proiectate ţinând cont de câteva condiţii:

- compatibilitate cu instrumentele software dezvoltate pe PC;
- accesibilitate deplină la toate facilităţile microcontrolerului;

- interconectare simplă cu un PC pentru a facilita transferul facil al programelor dezvoltate;
- simplitatea schemei necesară pentru a permite utilizatorului adăugarea numai a circuitelor strict necesare aplicaţiei.

În concluzie, sistemul de dezvoltare permite controlul total, prin intermediul programelor compilate pe un PC, asupra funcţionării microcontrolerului, lăsând şi posibilitatea utilizatorului de a adăuga circuite specifice (convertoare DAC, circuite I^2C, ceas de timp real, precum şi alte periferice).

2.2.1 Microcontrolerul 80C552

Sistemul de dezvoltare propus pentru circuitul 80C552 poate fi realizat fizic pe o placă de circuit EUROCARD, la care aproximativ 1/3 din suprafaţa plăcii poate fi rezervată utilizatorului pentru adăugarea de circuite suplimentare.

Principalele facilităţi ale schemei prezentate în figura 3.1 sunt:

- sursă de alimentare de 5 V şi 100 mA;
- este prevăzută o protecţie pentru prevenirea alimentării inverse a sistemului de dezvoltare;
- sistemul de dezvoltare are prevăzute jumpere pentru selectarea tipului de circuit utilizat (80C…, 83C… sau 87C…) precum şi pentru selectarea memoriei ROM (interne sau externe – semnalul EA);
- schimbul de date cu PC-ul este realizat implicit prin intermediul unei interfeţe RS-232 cu ASC0 în modul 1 cu caracteristicile:
  - viteză: 2400 baud;
  - număr de biţi date: 8;
  - fără paritate;
  - un bit de stop.
- memorie externă RAM pentru încărcare programe până la 32k × 8;
- memorie externă ROM pentru programul monitor până la 32k × 8;
- are suportul realizat pentru adăugarea de circuite I^2C şi display LCD;
- decodificatorul U5 – HCT138 permite selectarea a 8 periferice externe, cu adresele S0…S7, respectiv în primii 16 kB ai spaţiului de memorie RAM, conform cu tabelul 3.11.

Tabelul 2.11

Selecţie	Adresa																Exemplu
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
S0	0	x	x	x	x	x	x	1	0	0	0	x	x	x	x	x	010xh
S1	0	x	x	x	x	x	x	1	0	0	1	x	x	x	x	x	012xh
S2	0	x	x	x	x	x	x	1	0	1	0	x	x	x	x	x	014xh
S3	0	x	x	x	x	x	x	1	0	1	1	x	x	x	x	x	016xh
S4	0	x	x	x	x	x	x	1	1	0	0	x	x	x	x	x	018xh
S5	0	x	x	x	x	x	x	1	1	0	1	x	x	x	x	x	01axh
S6	0	x	x	x	x	x	x	1	1	1	0	x	x	x	x	x	01cxh
S7	0	x	x	x	x	x	x	1	1	1	1	x	x	x	x	x	01exh

- utilizarea unui circuit special destinat interfeţei RS-232, MAX232, dă posibilitatea folosirii tensiunilor bipolare nestabilizate produse de acesta

(aproximativ ±10V) pentru alimentarea câtorva circuite analogice cu condiţia de a avea un consum foarte mic şi fără a avea pretenţia unei calităţi bune a tensiunii de alimentare.

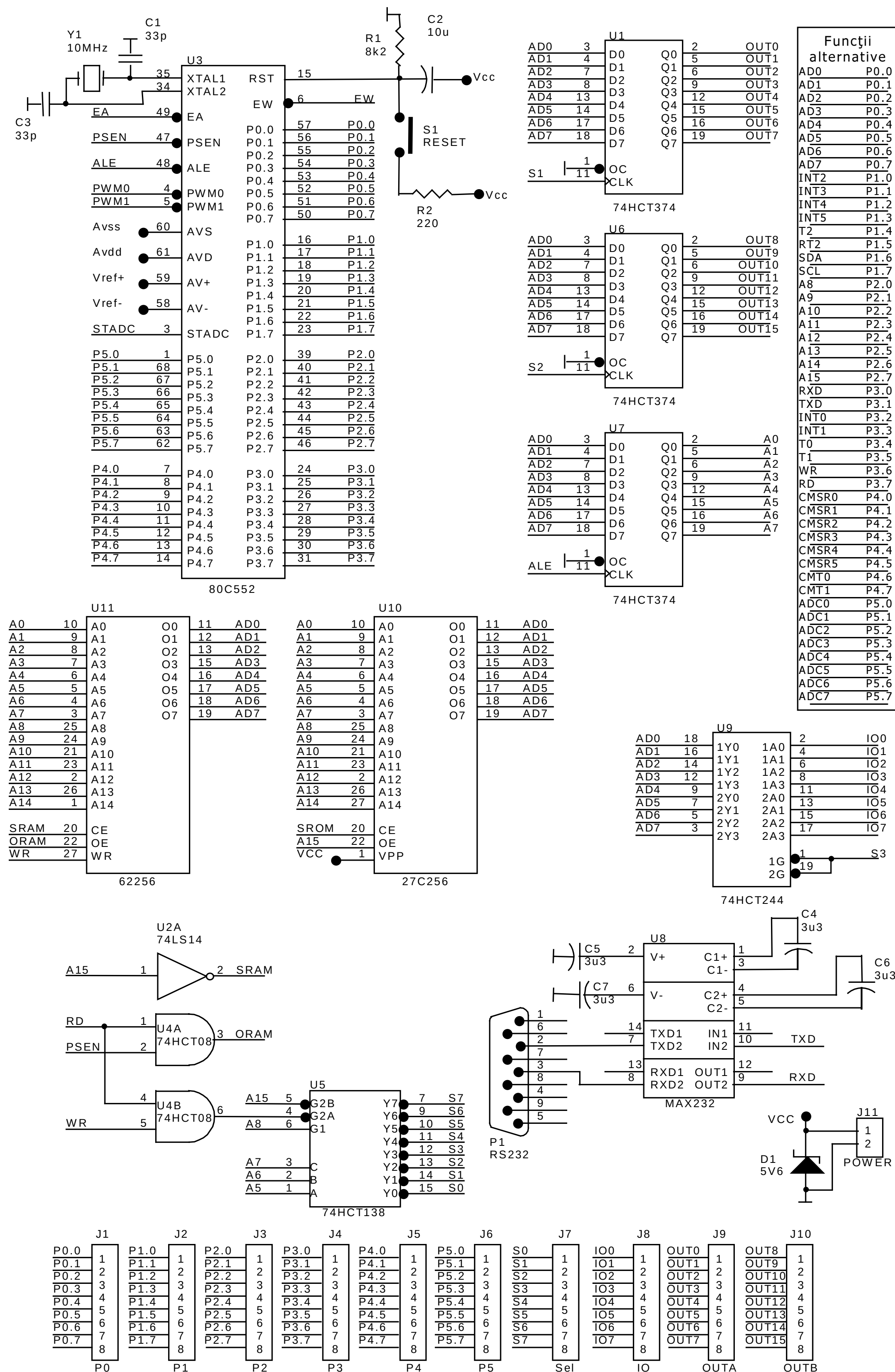


Figura 2.2. Sistem de dezvoltare cu 80C552

2.2.2 Microcontrolerul 80C167

Sistemul de dezvoltare cu microcontrolerul 80C167 permite dezvoltarea, depanarea şi execuţia programelor de aplicaţie realizate pe un PC. Schema electrică de principiu a sistemului este prezentată în figura 3.2.

Principalele facilităţi ale sistemului de dezvoltare sunt:

- selectarea memoriei externe, ROM+RAM sau numai RAM, precum şi a mărimii memoriei externe 1MB sau 2/4MB prin intermediul jumperelor JP1…JP4, respectiv JP6 şi JP7, după cum urmează:
  - JP1 şi JP2 pe poziţie "WRx" selectează U4 şi U5 ca RAM, iar pe poziţie "A15" selectează ca memorie ROM;
  - JP3 şi JP4 pe poziţie "A16/A15" selectează RAM, iar pe poziţie "A19" selectează ROM;
  - JP6 pe poziţie "V_{CC}" selectează memorie de 1 MB, iar pe poziţie "A18" selectează memorie de 2/4 MB;
  - JP7 pe poziţie "A19" selectează RAM, iar pe poziţie "V_{CC}" selectează memorie ROM.
- JP5 în poziţia "V_{CC}" selectează magistrală demultiplexată, iar în poziţia "GND" selectează magistrală multiplexată. Dacă este selectată magistrala multiplexată, utilizatorul trebuie să întrerupă semnalele AD1…AD15 între microcontroler şi circuitele de memorie U2 şi U3.
- setarea registrelor SYSCON, BUSCON0 şi RP0H prin intermediul jumperelor JP8…JP15, după cum urmează:
  - JP8 setează bitul WRCFG (SYSCON.7 – modul semnalelor $\overline{WR}$ şi $\overline{BHE}$);
  - JP9 validează încărcătorul bootstrap;
  - JP10 şi JP11 setează câmpul BTYP din registrul BUSCON0 care are următoarea semnificaţie:

JP10 (P0.6)	JP11 (P0.7)	Mod lucru EBC
OFF	OFF	Magistrală 16 biţi multiplexată
OFF	ON	Magistrală 16 biţi demultiplexată
ON	OFF	Magistrală 8 biţi multiplexată
ON	ON	Magistrală 8 biţi demultiplexată

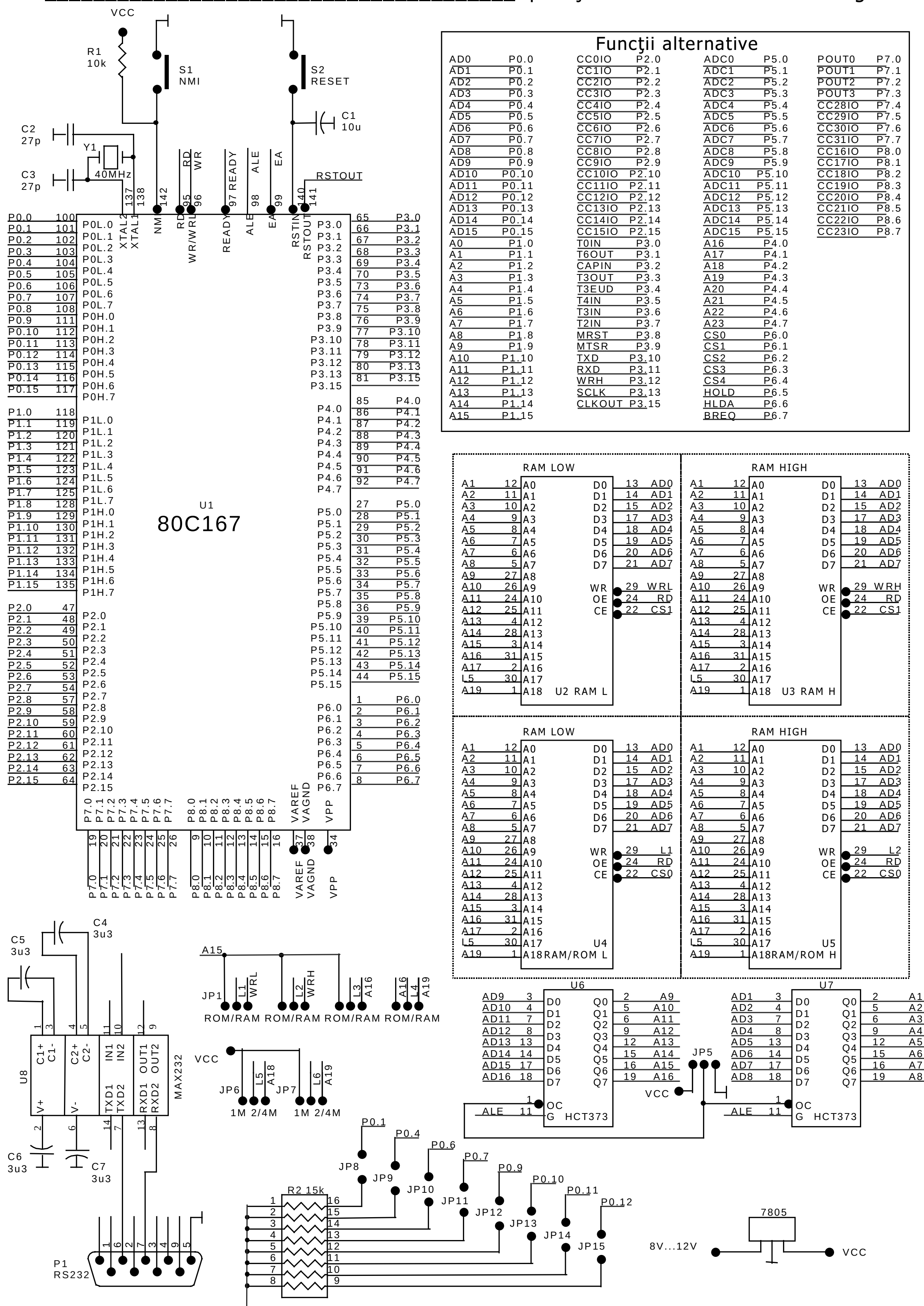
  - JP12 şi JP13 setează câmpul CSSEL din registrul RP0H care are următoarea semnificaţie:

JP12 (P0.9)	JP13 (P0.10)	Număr linii selectare
OFF	OFF	5 linii $\overline{CS0}$…$\overline{CS4}$
OFF	ON	Nici o linie $\overline{CS}$
ON	OFF	2 linii $\overline{CS0}$ şi $\overline{CS1}$
ON	ON	3 linii $\overline{CS0}$…$\overline{CS2}$

  - JP14 şi JP15 setează câmpul SALSEL din registrul RP0H care are următoarea semnificaţie:

JP14 (P0.11)	JP15 (P0.12)	Număr linii adresă segment
OFF	OFF	Adresă segment 2 biţi (A16…A17)
OFF	ON	Adresă segment 8 biţi (A16…A23)
ON	OFF	Memorie nesegmentată
ON	ON	Adresă segment 4 biţi (A16…A19)

- comunicarea între microcontroler şi calculator se face printr-o interfaţă serială asincronă RS-232;
- placa dispune de un buton pentru generarea unei întreruperi nemascabile $\overline{NMI}$ (S1) şi unul pentru reset extern (S2);
- alimentarea sistemului se face prin intermediul unei surse stabilizate LM7805 (LM323) cu o tensiune 8V…12V la un curent de maxim 300 mA;



Funcţii alternative

AD0	P0.0	CC0IO	P2.0	ADC0	P5.0	POUT0	P7.0
AD1	P0.1	CC1IO	P2.1	ADC1	P5.1	POUT1	P7.1
AD2	P0.2	CC2IO	P2.2	ADC2	P5.2	POUT2	P7.2
AD3	P0.3	CC3IO	P2.3	ADC3	P5.3	POUT3	P7.3
AD4	P0.4	CC4IO	P2.4	ADC4	P5.4	CC28IO	P7.4
AD5	P0.5	CC5IO	P2.5	ADC5	P5.5	CC29IO	P7.5
AD6	P0.6	CC6IO	P2.6	ADC6	P5.6	CC30IO	P7.6
AD7	P0.7	CC7IO	P2.7	ADC7	P5.7	CC31IO	P7.7
AD8	P0.8	CC8IO	P2.8	ADC8	P5.8	CC16IO	P8.0
AD9	P0.9	CC9IO	P2.9	ADC9	P5.9	CC17IO	P8.1
AD10	P0.10	CC10IO	P2.10	ADC10	P5.10	CC18IO	P8.2
AD11	P0.11	CC11IO	P2.11	ADC11	P5.11	CC19IO	P8.3
AD12	P0.12	CC12IO	P2.12	ADC12	P5.12	CC20IO	P8.4
AD13	P0.13	CC13IO	P2.13	ADC13	P5.13	CC21IO	P8.5
AD14	P0.14	CC14IO	P2.14	ADC14	P5.14	CC22IO	P8.6
AD15	P0.15	CC15IO	P2.15	ADC15	P5.15	CC23IO	P8.7
A0	P1.0	T0IN	P3.0	A16	P4.0		
A1	P1.1	T6OUT	P3.1	A17	P4.1		
A2	P1.2	CAPIN	P3.2	A18	P4.2		
A3	P1.3	T3OUT	P3.3	A19	P4.3		
A4	P1.4	T3EUD	P3.4	A20	P4.4		
A5	P1.5	T4IN	P3.5	A21	P4.5		
A6	P1.6	T3IN	P3.6	A22	P4.6		
A7	P1.7	T2IN	P3.7	A23	P4.7		
A8	P1.8	MRST	P3.8	CS0	P6.0		
A9	P1.9	MTSR	P3.9	CS1	P6.1		
A10	P1.10	TXD	P3.10	CS2	P6.2		
A11	P1.11	RXD	P3.11	CS3	P6.3		
A12	P1.12	WRH	P3.12	CS4	P6.4		
A13	P1.13	SCLK	P3.13	HOLD	P6.5		
A14	P1.14	CLKOUT	P3.15	HLDA	P6.6		
A15	P1.15			BREQ	P6.7		

Figura 2.3. Sistem de dezvoltare cu 80C167

- tot sistemul de dezvoltare este realizabil pe o placă EUROCARD, din care aproximativ 1/3 din suprafaţa plăcii poate fi rezervată utilizatorului pentru adăugarea de circuite suplimentare.

2.3. Afişarea informaţiilor

Multe din dispozitivele controlate de microcontrolere, necesită unele elemente de semnalizare pentru verificarea stării sistemului. Dacă în unele situaţii sunt suficiente câteva LED-uri sau digiţi cu 7 segmente, acestea putând fi comandate direct de porturile de intrare/ieşire, unele aplicaţii pot necesita afişarea unor informaţii mai complexe, cum ar fi caractere alfa-numerice, caractere speciale şi chiar imagini vectoriale (drepte, cercuri, pătrate etc.).

Trebuie menţionat că, deoarece porturile microcontrolerului sunt qvasi bidirecţionale, LED-ul trebuie conectat între V_{CC} şi pinul portului prin intermediul unei rezistenţe de 470 sau 520 ohmi.

Dintre nenumăratele posibilităţi pentru afişarea informaţiilor, vor fi prezentate numai dispozitivele de afişare pe tub catodic şi cele cu display alfanumeric LCD.

2.3.1 Afişarea pe tub catodic

Principiul funcţionării acestor dispozitive se bazează pe producerea, de către microcontroler, a două semnale analogice pentru comanda deplasării spotului tubului catodic pe orizontală şi verticală (X şi Y), precum şi a unui semnal digital pentru comanda amplificatorului video (Z).

Pentru un amator, realizarea circuitelor suport pentru un tub catodic (surse de înaltă tensiune, amplificatoare video de viteză foarte mare etc.) pot fi prohibitive. Totuşi, pentru aplicaţii de laborator, se poate utiliza un osciloscop catodic, setat corespunzător, care preia semnalele microcontrolerului pentru a le afişa.

O schemă bloc de realizare a unui asemenea dispozitiv este prezentată în figura 3.3.

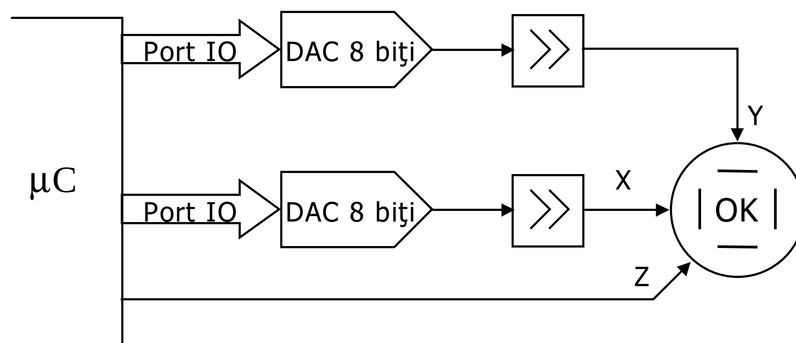


Figura 2.4. Dispozitiv de afişare pe osciloscop

În continuare, este prezentat un program, în asamblor pentru 80C552, care poate fi folosit pentru afişarea unor caractere alfanumerice recepţionate pe interfaţa serială.

Caracterele sunt generate prin intermediul unor matrice de 8×8 pixeli. Utilizatorul poate adăuga alte caractere introducând în tabel valorile dorite.

```
/***********************************************************************\
```

```
**  Descriere:    Rutină afişare caractere pe osciloscop                 **
**                                                                       **
**  Versiune:     1.0                                                    **
**  Inceut la:    Iunie 94                                               **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania     **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                     **
**                                                                       **
**  Acest modul contine urmatoarele functii:                             **
**                                                                       **
**  Ultima modificare la data: 23 nov 1998                               **
**  Istoric:                                                             **
**                                                                       **
**  Observatii:                                                          **
**                                                                       **
\*************************************************************************/
ch_o       data      30h                ; date DAC orizontal
ch_v       data      31h                ; date DAC vertical
oriz       data      32h                ; pozitie inceput caracter x
vert       data      33h                ; pozitie inceput caracter y

           ; Setari initiale (pozitie reticul)
           mov       oriz,#79h          ;iniţializare inceput text
           mov       vert,#79h          ;

setport:
           mov       p2,#1              ;acesam porturile cu adresele
           mov       r0,#20h            ;0x0120 (DAC orizontal) si
           mov       r1,#40h            ;0x0140 (DAC vertical)

           mov       DPTR,#txt          ; citim textul de afisat
nextchr:   movc      A,@DPTR            ; citim un caracter
           jz        gata               ; este 0 -> gata

           inc       DPTR               ; ptr. caracter urmator
           push      DPL                ; salvam ptr caracter
           push      DPH

           mov       DPTR,#gen_c        ; ptr. generattor caractere

           clr       C                  ; stergem CY pt. subb
           subb      A,#20h             ; scadem offset tabel pt. ASCII

           mov       B,#8               ; calculam ptr. in tabelul cu
           mul       AB                 ; caratere in functie de caracterul
           add       A,DPL              ; care trebuie sa fie afisat
           mov       DPL,A              ; DPTR + chr*8
           mov       B,A                ; un carcter are 8 coloane si
           adc       A,DPH              ; 8 randuri
           mov       DPH,A

           mov       r2,#7              ; contor randuri
           mov       r3,#7              ; contor coloane

           mov       ch_v,vert          ; initializare DAC vertical

nextlin:   movc      A,@DPTR            ; citim un rand
           mov       ch_o,oriz          ; DAC la inceput caracter
nextpixel: RRC       A                  ; testare pixel de afisat
           mov       B,A                ; salvare rand
           jnc       endpixel           ; pixelul nu exista
           call      afis               ; desenam pixelul
endpixel:  inc       ch_o               ; urmatoarea coloana
           mov       A,B                ; refacere valoare rand
           djnz      r3,nextpixel       ; urmatorul rand

           inc       ch_v               ; DAC la urmatorul rand
           inc       DPTR               ; ptr. la urmatorul rand
           djnz      r2,nextlin         ; salt la urmatoarea linie

           mov       oriz,ch_o          ; setare poz. inceput caracter urm.

           pop       DPH                ; refacere DPTR cu ptr. caracter urm.
           POP       DPL
           jmp       nextchr            ; desenam urmatorul caracter
gata:      ret

afis:      mov       a,ch_o             ;afisare pixeli
           movx      @r0,a
           mov       a,ch_v
           movx      @r1,a              ;aşteptare 1 us
           nop
           nop
           nop
```

```
          nop
          nop
          clr       p4.2             ;set z
          nop                        ;aşteptare 1 us
          nop
          nop
          nop
          nop
          setb      p4.2             ;reset z
          ret

txt:      db        "Un text", 0

gen_c:    db        0,0,0,0,0,0,0,0                   ;blanc
          db        20h,20h,20h,20h,0,0,20h,0         ;!
          db        50h,50h,50h,0,0,0,0,0             ;"
          db        50h,50h,0f8h,50h,0f8h,50h,50h,0   ;#
          db        20h,78h,0a0h,70h,28h,0f0h,20h,0   ;$
          db        0c0h,0c8h,10h,20h,40h,98h,18h,0   ;%
          db        60h,90h,0a0h,40h,0a8h,90h,68h,0   ;&
          db        60h,20h,40h,0,0,0,0,0             ;'
          db        10h,20h,40h,40h,40h,20h,10h,0     ;(
          db        40h,20h,10h,10h,10h,20h,40h,0     ;)
          db        0,20h,0a8h,70h,0a8h,20h,0,0       ;*
          db        0,20h,20h,0f8h,20h,20h,0,0        ;+
          db        0,0,0,0,60h,20h,40h,0             ;,
          db        0,0,0,0f8h,0,0,0,0                ;-
          db        0,0,0,0,0,60h,60h,0               ;.
          db        0,8,10h,20h,40h,80h,0,0           ;/
          db        70h,88h,98h,0a8h,0c8h,88h,70h,0   ;0
          db        20h,60h,20h,20h,20h,20h,70h,0     ;1
          db        70h,88h,8,10h,20h,40h,0f8h,0      ;2
          db        0f8h,10h,20h,10h,8,88h,0f0h,0     ;3
          db        10h,30h,50h,90h,0f8h,10h,10h,0    ;4
          db        0f8h,80h,0f0h,8,8,88h,70h,0       ;5
          db        30h,40h,80h,0f0h,88h,88h,70h,0    ;6
          db        0f8h,8,10h,20h,40h,40h,40h,0      ;7
          db        70h,88h,88h,70h,88h,88h,70h,0     ;8
          db        70h,88h,88h,78h,8,10h,60h,0       ;9
          db        0,60h,60h,0,60h,60h,0,0           ;:
          db        0,60h,60h,0,60h,20h,40h,0         ;;
          db        10h,20h,40h,80h,40h,20h,10h,0     ;<
          db        0,0,0f8h,0,0f8h,0,0,0             ;=
          db        80h,40h,20h,10h,20h,40h,80h,0     ;>
          db        70h,88h,8,10h,20h,0,20h,0         ;?
          db        70h,88h,8,68h,0a8h,0a8h,70h,0     ;@
          db        70h,88h,88h,88h,0f8h,88h,88h,0    ;A
          db        0f0h,88h,88h,0f0h,88h,88h,0f0h,0  ;B
          db        70h,88h,80h,80h,80h,88h,70h,0     ;C
          db        0e0h,90h,88h,88h,88h,90h,0e0h,0   ;D
          db        0f8h,80h,80h,0f0h,80h,80h,0f8h,0  ;E
          db        0f8h,80h,80h,0f0h,80h,80h,80h,0   ;F
          db        70h,88h,80h,0b8h,88h,88h,78h,0    ;G
          db        88h,88h,88h,0f8h,88h,88h,88h,0    ;H
          db        70h,20h,20h,20h,20h,20h,70h,0     ;I
          db        38h,10h,10h,10h,10h,90h,60h,0     ;J
          db        88h,90h,0a0h,0c0h,0a0h,90h,88h,0  ;K
          db        80h,80h,80h,80h,80h,80h,0f8h,0    ;L
          db        88h,0d8h,0a8h,0a8h,88h,88h,88h,0  ;M
          db        88h,88h,0c8h,0a8h,98h,88h,88h,0   ;N
          db        70h,88h,88h,88h,88h,88h,70h,0     ;O
          db        0f0h,88h,88h,0f0h,80h,80h,80h,0   ;P
          db        70h,88h,88h,88h,0a8h,90h,68h,0    ;Q
          db        0f0h,88h,88h,0f0h,0a0h,90h,88h,0  ;R
          db        78h,80h,80h,70h,8,8,0f0h,0        ;S
          db        0f8h,20h,20h,20h,20h,20h,20h,0    ;T
          db        88h,88h,88h,88h,88h,88h,70h,0     ;U
          db        88h,88h,88h,88h,88h,50h,20h,0     ;V
          db        88h,88h,88h,0a8h,0a8h,0a8h,50h,0  ;W
          db        88h,88h,50h,20h,50h,88h,88h,0     ;X
          db        88h,88h,88h,50h,20h,20h,20h,0     ;Y
          db        0f8h,8,10h,20h,40h,80h,0f8h,0     ;Z
          db        70h,40h,40h,40h,40h,40h,70h,0     ;[
          db        0,80h,40h,20h,10h,8,0,0           ;\
          db        70h,10h,10h,10h,10h,10h,70h,0     ;]
          db        20h,50h,88h,0,0,0,0,0             ;^
          db        0,0,0,0,0,0,0f8h,0                ;_
          db        0,20h,20h,0f8h,20h,20h,0,0        ;+
```

Folosirea osciloscopului prezintă o serie de avantaje, în sensul că, programând convenabil tensiunile de comandă U_X şi U_Y, ecranul poate fi transformat în display grafic putând fi generate chiar imagini simple, create

din segmente de linii. Un astfel de program pentru *desenarea* unui reticul la coordonate X,Y este prezentat în continuare.

```
/***********************************************************************\
**  Descriere:    Rutină afişare vectori pe osciloscop                **
**                                                                     **
**  Versiune:     1.0                                                  **
**  Inceut la:    Iunie 94                                             **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania   **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                   **
**                                                                     **
**  Acest modul contine urmatoarele functii:                           **
**                                                                     **
**  Ultima modificare la data: 23 nov 1998                             **
**  Istoric:                                                           **
**                                                                     **
**  Observatii:                                                        **
**                                                                     **
\***********************************************************************/
ch_o      data      30h                ;orizontală
ch_v      data      31h                ;verticală
oriz      data      32h
vert      data      33h
dep_sd    data      34h
dep_sj    data      35h

          ; Setari initiale (pozitie reticul)
          mov       oriz,#79h          ;iniţializare repere, (oriz, vert)
          mov       vert,#79h          ;este pozitie cetrului reticulului
          mov       dep_sd,#79h
          mov       dep_sj,#7eh
reticul:
          mov       p2,#1              ;acesam porturile cu adresele
          mov       r0,#20h            ;0x0120 (DAC orizontal) si
          mov       r1,#40h            ;0x0140 (DAC vertical)

sub_r:    mov       a,oriz             ;afisare reticul
          mov       ch_v,vert          ;valoare DAC veritical
          add       a,#7
          mov       ch_o,a
          movx      @r0,a              ;comandam DAC-ul orizontal
          mov       a,ch_v
          movx      @r1,a              ;comandam DAC-ul vertical
          mov       r7,#50h            ;asteptam un timp pt. desenare
wait1:    nop
          djnz      r7,wait1           ;aşteptare
          acall     afis               ;afisam un pixel

          mov       r7,#0eh            ;desenam linia veritcala
ret0:     inc       ch_v               ;ne deplasam pe verticala
          acall     afis               ;desenam un pixel la (ch_o, ch_v)
          djnz      r7,ret0

          mov       ch_o,oriz          ;
          mov       a,vert
          add       a,#7
          mov       ch_v,a
          movx      @r1,a
          mov       a,ch_o
          movx      @r0,a
          mov       r7,#20h
wait2:    nop
          djnz      r7,wait2
          acall     afis

          mov       r7,#0eh            ;deseneam linia orizontlala
ret1:     inc       ch_o
          acall     afis
          djnz      r7,ret1

          ret

afis:     mov       a,ch_o             ;afisare pixeli
          movx      @r0,a
          mov       a,ch_v
          movx      @r1,a              ;aşteptare 1 us
          nop
          nop
          nop
          nop
          nop
          clr       p4.2               ;set z
```

```
        nop                     ;aşteptare 1 us
        nop
        nop
        nop
        nop
        setb    p4.2            ;reset z
        ret
```

2.3.2 Afişarea pe display LCD

Afişoarele LCD au căpătat o foarte mare dezvoltare, fiind folosite în special ca dispozitive pentru afişarea informaţiilor pentru sistemele cu microcontrolere.

Dacă iniţial erau dispozitive cu 7 segmente, utilizabile în special pentru afişarea numerelor, astăzi s-au generalizat circuitele cu matrice de puncte organizate în 2…4 linii de caractere, fiecare linie conţinând 16…40 de caractere alfanumerice. Există şi afişoare LCD grafice proiectate pentru utilizare cu microcontrolere. Acestea nu sunt organizate ca matrice de caractere ci, mai degrabă, ca un display, existând rezoluţii de la 190x190 până la 640x480 puncte.

Avantajele dispozitivelor LCD constau în consumul extrem de redus, gabaritul mic, contrastul foarte bun şi vizibilitatea bună în condiţii de iluminare puternică.

Dezavantajele afişoarelor, circuitele de comandă complexe, tensiuni de alimentare diferite de restul circuitelor şi necesitatea unei iluminări externe în condiţii de întuneric, au fost în bună parte eliminate de modulele actuale. Acestea conţin un microcontroler specializat care are ca sarcină interfaţarea cu sistemul master, decodificarea datelor, controlul memoriei de caractere, controlul tensiunilor de alimentare etc.

Astăzi există pe piaţă o multitudine de dispozitive, în mare parte fiind echivalente între ele, în continuare fiind prezentate structura pinilor, comenzile şi un exemplu de program sursă pentru un display cu 2 rânduri şi 16 caractere.

Programele sursă folosesc fişierele de definiţii `SYSTEM.H` şi `TYPEDEF.H` care sunt prezentate în anexe.

În tabelul 3.12 sunt prezentate funcţiile pinilor modulului afişor LCD iar în tabelul 3.13 este prezentat registrul de comenzi şi setul de instrucţiuni al controlerului afişorului.

Tabelul 2.12

Pin nr.	Simbol	Nivel	Descriere	Funcţie	
1	V_{SS}	---	Masă	0V	
2	V_{DD}	---	Alimentare	5V±5%	
3	V_0		Ajustare contrast	0V(max)÷5V	
4	RS	H/L	Selectare registru	H: Data input L: Instruction code input	
5	R/$\overline{W}$	H/L	Citire/Scriere	H: Citire, L: Scriere	
6	E	H, H→L	Selecţie afişaj (Enable)	---	
7	DB0	H/L	Bitul 0 de date		mod 8 biţi
8	DB1	H/L	Bitul 1 de date		
9	DB2	H/L	Bitul 2 de date		
10	DB3	H/L	Bitul 3 de date		

11	DB4	H/L	Bitul 4 de date	mod 4 biţi	
12	DB5	H/L	Bitul 5 de date		
13	DB6	H/L	Bitul 6 de date		
14	DB7	H/L	Bitul 7 de date		

Tabelul 2.13

Instrucţiunea	Codul										Funcţia	Timpul de execuţie f_{OSC}=250kHz
	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Display Clear	0	0	0	0	0	0	0	0	0	1	Şterge toată zona de date, reface stara afişajului, încarcă contorul de adrese cu adresa DD RAM 00h	1.64ms
Display/ Cursor Home	0	0	0	0	0	0	0	0	1	*	Reface starea afişajului din deplasare şi incarcă contorul de adrese cu adresa DD RAM 00h	1.64ms
Entry mode set	0	0	0	0	0	0	0	1	I/$\overline{D}$	S	Indică direcţia de deplasare a cursorului şi a deplasarea afişajului. Această operaţie are loc după fiecare dată transferată.	40µs
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	Indică şi activează afişajul (D), cursorul (C), pâlpâirea caracterului aflat la poziţia cursorului (B).	40µs
Display/ Cursor Shift	0	0	0	0	0	1	S/$\overline{C}$	R/$\overline{L}$	*	*	Deplasare afişaj (S/$\overline{C}$=1), sau cursor (S/$\overline{C}$=0) la dreapta (R/$\overline{L}$=1) sau stânga (R/$\overline{L}$=0).	40µs
Function Set	0	0	0	0	1	DL	N	0	*	*	Setează numărul de biţi ai interfeţei (DL=1: 8 biţi, DL=0: 4 biţi) şi numărul de linii ai afişajului (N=1: două linii, N=0: o linie).	40µs
CG RAM Address Set	0	0	0	1	ACG						Încarcă contorul de adrese cu o adresă CG RAM. Toate datele ulterioare vor fi din/în CG RAM.	40µs
DD RAM Address Set	0	0	1	ADD							Încarcă contorul de adrese cu o adresă DD RAM. Toate datele ulterioare vor fi din/în DD RAM.	40µs
Busy Flag / Address Counter read	0	1	BF	AC							Citire indicatorului BUSY (BF) şi a contorului de adrese (AC).	40µs
CG RAM / DD RAM Data Write	1	0	Datele ce vor fi înscrise								Scrie date în CG RAM sau DD RAM.	40µs
CG RAM / DD RAM Data Write	1	1	Datele care sunt citite								Citeşte date din CG RAM sau DD RAM.	40µs

```
/*************************************************************************\
**  Titlu:          LCD.H                                                **
**  Descriere:      Declaratiile functiilor pentru lucru cu LCD-ul       **
**                                                                       **
**  Versiune:       2.0                                                  **
**  Inceut la:      August 97                                            **
**  Autor:          Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania   **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                     **
**                                                                       **
**  Acest modul contine urmatoarele functii:                             **
**                  declaratii de coduri si macrouri                     **
```

```
**                                                                      **
**  Ultima modificare la data: 23 nov 1998                              **
**  Istoric:                                                            **
**                                                                      **
**  Observatii:                                                         **
**                                                                      **
\************************************************************************/
#ifndef _LCD_H_
#define _LCD_H_         1
#include <typedef.h>

//#define _LCD_BUFFER_  1                     /* validare lucru bufferat */

extern void LCDcmd(char c);                   // scrie o comanda la LCD
extern void LCDdta(char d);                   // scrie date la LCD
extern void LCD_init(void);                   // initializare LCD
extern void LCDclear(bit row);                // sterge LCD (buffer LCD)
extern void LCDclr(bit row);                  // sterge LCD (sirect la LCD)
extern void LCDstr(bit rand, char col, char* txt);     // scrie txt la rand,col
extern void LCDchr(bit rand, char col, char chr);      // scrie chr la rand,col

#define CursorON()      LCDcmd(0x0E)
#define CursorOFF()     LCDcmd(0x0C)
#define BlinkON()       LCDcmd(0x0D)
#define BlinkOFF()      LCDcmd(0x0C)
#define BonCoff()       LCDcmd(0x0D)     /* Blink ON Cursor OFF */
#define BoffCon()       LCDcmd(0x0E)     /* Blink OFF Cursor ON */

bit CursorRight(void);                   // 1=EOL
bit CursorLeft(void);                    // 1=BOL
byte LCDgetpoz(void);                    // 0x00-0x17 rind 1, 0x40-0x57 rind 2
void LCDsetpoz(byte poz);                // 0x00-0x17 rind 1, 0x40-0x57 rind 2

#ifdef _LCD_BUFFER_
  #include "../MAIN/system.h"
  extern xdata char LCDTXT[2][LCD_SIZE+1]; /* imaginea in memorie
                                   a LCD-ului */
  extern bit fLCD;              /* Flag date noi pt LCD                */
                                /* Indica ca sunt daate noi de afisat in
                                   bufferul LCDTXT                     */
  extern bit fLCDon;            /* Flag afisare permisa LCD            */
                                /* fLCDon = 0 blocheaza afisarea pe LCD
                                   este folosit pentru a bloca afisarea
                                   cand se folosesc functii lungi (printf)
                                   si care pot lasa bufferul LCDTXT
                                   intr-o stare nedefinita (nu ASCIIZ)  */
  extern void LCDput(bit rand, char col, char *txt);
#endif

#endif

/************************************************************************\
**  Titlu:        LCD.C                                                 **
**  Descriere:    Functiile folosite pentru acesul la LCD               **
\************************************************************************/
#include <reg552.h>
#include <stdio.h>
#include <string.h>
#include <absacc.h>
#include <typedef.h>
#include "..\main\system.h"
#include "lcd.h"

#ifdef _LCD_BUFFER_
  xdata char    LCDTXT[2][LCD_SIZE+1];
  bit   fLCD;           // Flag date noi pentru LCD
  bit   fLCDon;         // Flag afisare permisa pe LCD
#endif

\***********************************************************************
LCDcmd = transmite o comanda la LCD
***********************************************************************/
void LCDcmd(char c)
{
  waitLCD();
  LCDwcmd(c);
}

\***********************************************************************
iLCDcmd = transmite o comanda la LCD. Folosita in intreruperi cand se
          foloseste bufferarea afisajului
***********************************************************************/
#ifdef _LCD_BUFFER_
```

```
void iLCDcmd(char c)
{
  waitLCD();
  LCDwcmd(c);
}
#endif

\**************************************************************************
LCDdata = transmite date LCD-ului
*************************************************************************/
void LCDdta(char d)
{
  waitLCD();
  LCDwdta(d);
}

/* ****************************************************************************
RS  R/W 7   6   5   4   3   2   1   0
0   0   0   0   0   0   0   0   0   1   Display Clear
0   0   0   0   0   0   0   0   1   x   Display/Cursor Home
0   0   0   0   0   0   0   1   I/D S   Cursor direction, Display Shift
0   0   0   0   0   0   1   D   C   B   D = Display ON/OFF
                                        C = Cursor ON/OFF
                                        B = Blink ON/OFF
0   0   0   0   0   1   S/C R/L x   x   Shift Display(1), Move Cursor(0)
                                        Shift Right(1), Shift Left(0)
0   0   0   0   1   DL  N   0   x   x   DL=1->8bit, DL=0->4bit
                                        N=1->Dual line, N=0->Single line
0   0   0   1   ------- ACG ---------   CG RAM Address Set
0   0   1   ----------- ACG ---------   DD RAM Address Set
0   1   BF  ----------- AC ----------   Busy Flag, Address Counter
1   0   ------- Write Data ----------   Write data to CG RAM or DD RAM
1   1   ------- Read Data -----------   Read data from CG RAM or DD RAM
*************************************************************************** */
\**************************************************************************
LCD_init = initilizare LCD
*************************************************************************/
void LCD_init(void)
{
#ifdef _LCD_BUFFER_
  iLCDcmd(0x38);  // function set.8 biti, 2 linii, 5x7 dot matrix
  iLCDcmd(0x06);  // entry mode set.Increment adr, no display shift
  iLCDcmd(0x0C);  // display ON.Dissplay on/off control,Cursor ON
  iLCDcmd(0x01);  // clear disp

  LCDclear(0);
  LCDclear(1);
#else
  LCDcmd(0x38);  // function set.8 biti, 2 linii, 5x7 dot matrix
  LCDcmd(0x06);  // entry mode set.Increment adr, no display shift
  LCDcmd(0x0C);  // display ON.Dissplay on/off control,Cursor ON
  LCDcmd(0x01);  // clear disp
#endif
}

\**************************************************************************
LCDgetpoz = intoarce pozitia unde se afla cursorul
*************************************************************************/
byte LCDgetpoz(void)
{
  data byte poz;

  do
    poz = XBYTE[0x0102];
  while (poz & 0x80);
  return (poz & 0x7f);
}

\**************************************************************************
LCDsetpoz = muta cursorul la pozitia specificata
*************************************************************************/
void LCDsetpoz(byte poz)
{
  waitLCD();
  LCDwcmd(poz | 0x80);
}

\**************************************************************************
CursorRight = muta cursorul la dreapta
*************************************************************************/
bit CursorRight(void)  // 1=EOL
{
  data byte poz;
```

```
  do
    poz = XBYTE[0x0102];
  while (poz & 0x80);

  if((poz == 0x17) || (poz == 0x57))
    return 1;
  LCDwcmd(++poz | 0x80);
  return 0;
}

\**************************************************************************
CursorLeft = muta cursorul la stanga
**************************************************************************/
bit CursorLeft(void)  // 1=BOL
{
  data byte poz;

  do
    poz = XBYTE[0x0102];
  while (poz & 0x80);

  if((poz == 0x00) || (poz == 0x40))
    return 1;
  LCDwcmd(--poz | 0x80);
  return 0;
}

\**************************************************************************
         scrie un text la randul si coloana specificata
LCDput = apelata atunci cand se foloseste dublu bufferarea
LCDstr = apelata atunci cand nu se foloseste dublu bufferarea
**************************************************************************/
#ifdef _LCD_BUFFER_
  void LCDput(bit rand, char col, char* txt)
#else
  void LCDstr(bit rand, char col, char* txt)
#endif
{
  data char i;

  // randul 0 incepe de la adresa 0x00 pana la 0x17 = 24 de pozitii
  // randul 1 incepe de la adresa 0x40 pana la 0x57

#ifdef _LCD_BUFFER_
  iLCDcmd(0x80 | (rand ? (0x40 + col) : col ));
#else
  LCDcmd(0x80 | (rand ? (0x40 + col) : col ));
#endif

  for(i=0; txt[i] && (i <= LCD_SIZE); i++)
     LCDdta(txt[i]);
/*
  for(i=0; txt[i] && (i <= LCD_SIZE); i++)
     if(txt[i] != '\n')
       LCDdta(txt[i]);
     else
       i--;
*/
}




\**************************************************************************
LCDclr = sterge afisajul la randul specificat. Scrie direct la LCD
**************************************************************************/
void LCDclr(bit row)
{
  LCDstr(row, 0, "                    ");
}

/*
void LCDchr(bit rand, char col, char chr)
{
  unsigned char i;

  // randul 0 incepe de la adresa 0x00 pana la 0x17 = 24 de pozitii
  // randul 1 incepe de la adresa 0x40 pana la 0x57
```

```
//  col = (col > 0x17) ? 0x17 : col;
//  col = (col < 0) ? 0 : col;

  LCDcmd(0x80 | (rand ? (0x40 + col) : col ));

  LCDdta(chr);
}
*/

#ifdef _LCD_BUFFER_
\**************************************************************************
LCDclear = sterge randul din imaginea textului LCD in memorie
**************************************************************************/
void LCDclear(bit row)
{
  byte r;
  r = row ? 1 : 0;

  fLCDon = 0;
  memset(LCDTXT[r], 0, LCD_SIZE+1);
  fLCD = fLCDon = 1;
}

\**************************************************************************
LCDstr = scrie un sir de caractere la randul si coloana specificate,
         direct la LCD
**************************************************************************/
void LCDstr(bit rand, char col, char *txt)
{
  xdata byte r,c,i;

  r = rand ? 1 : 0;
  c = 0;

  while(LCDTXT[r][c] && (c < LCD_SIZE)) // cautam ASCIZ
    c++;

  fLCDon = 0;           // Inhibam afisarea pe LCD

  if(c < col)           // copletam cu spatii albe pina la colana
  {
    for( ;c < col; c++)
       LCDTXT[r][c] = ' ';
    LCDTXT[r][c] = 0;
  }

  // copiem textul
  for(c = col,i=0; txt[i] && (c < LCD_SIZE) ; c++, i++)
     LCDTXT[r][c] = txt[i];
  LCDTXT[r][c] = 0;

  // Validam afisarea, marcam ca exista date noi de afisat
  fLCD = fLCDon = 1;
}

#endif
```

2.4. Tastatură matricială

Pentru citirea tastaturii se foloseşte portul P4 al microcontrolerului.

Programul expus în continuare permite citirea atât a tastaturilor 3x4 cât şi a tastaturilor 4x4. Forma acestor tastaturi şi decodarea lor este prezentată în figura de mai jos. Legarea tastaturii la portul P4 s-a făcut în aşa fel încât să se obţină acelaşi cod pentru aceeaşi tastă atât în cazul folosirii tastaturii 3x4 cât şi a tastaturii 4x4; deoarece citirea tastaturii foloseşte logica negată sunt necesare rezistenţe la V_{CC} la fiecare pin al portului P4. Modul de conectare a tastaturii este prezentat în figura 3.5.

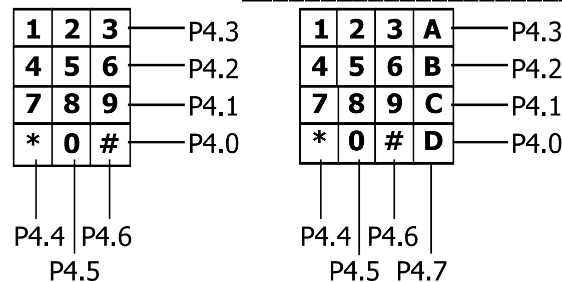


Figura 2.5. Conectarea tastaturii la sistemele cu 80C552

2.4.1 Rutine pentru utilizarea tastaturii pe sistemele cu 80C552

Citirea tastaturii se face folosind următorul algoritm:

Pasul 1. Se emit pe rânduri (P4.0÷P4.3) 0000 şi se setează coloanele ca intrări (P4.4÷P4.7) 1111;

Pasul 2. Se citesc coloanele (P4.4÷P4.7);

Pasul 3. Se emite pe colane ce s-a citit şi se setează rândurile ca intrări (P4.0÷P4.3) 1111;

Pasul 4. Se citeşte şi complementează valoarea portului P4.

Pasul 5. Dacă valoarea citită este diferită de zero, atunci este apăsată o tastă. Astfel se obţine un cod unic la apăsarea unei taste şi de asemenea se pot obţine şi coduri unice la apăsarea unor combinaţii de taste.

Codurile obţinute la tastatură sunt interpretate folosind declaraţiile din fişierul KBRDCODE.H.

Pentru iniţializarea tastaturii se apelează funcţia KBRD_init care setează portul P4 şi, dacă se foloseşte bufferul de tastatură, se iniţializează pointerii de citire şi scriere.

Folosirea bufferului pentru citirea tastaturii este împărţită în două funcţii.

Prima funcţie iKBRD_read este apelată în întreruperi şi depune codurile tastelor apăsate într-o stivă implementată cu ajutorul unui buffer cu adresare circulară. Citirea din această stivă, se face acolo unde este necesar folosind o a doua funcţie KBRD_read. Indicarea apăsării unei taste se face prin indicatorul KBRDHIT care se poate verifica înainte de apelul funcţiei KBRD_read.

În cazul în care nu se foloseşte bufferul de tastatură se apelează numai funcţia KBRD_read care întoarce unul din codurile de eroare sau codul tastei apăsate.

Nici una din funcţii nu aşteaptă ca să fie apăsata o tastă şi deci trebuie apelate în polling.

Programele sursă folosesc fişierele de definiţii SYSTEM.H şi TYPEDEF.H care sunt prezentate în anexe.

```
/*********************************************************************\
**  Titlu:        KBRDCODE.H                                        **
**  Descriere:    Codurile returnate de KBRDread() la citirea tastaturii  **
**                                                                  **
**  Versiune:     2.0                                               **
**  Început la:   August 97                                         **
**  Autor:        Ştefan Suceveanu, Pratco s.r.l. Bucureşti, România      **
```

```
**  Compilator C: C51 V2.27, Franklin Software, Inc.                     **
**                                                                       **
**  Acest modul conţine următoarele funcţii:                             **
**              declaraţii de coduri                                     **
**                                                                       **
**  Ultima modificare la data: 23 nov 1998                               **
**  Istoric:                                                             **
**                                                                       **
\*************************************************************************/
#ifndef      _KBRDCODE_H_
#define      _KBRDCODE_H_

// Tastatura 3x4 si/sau 4x4
#define      Kb1   0x18
#define      Kb2   0x28
#define      Kb3   0x48
#define      Kb4   0x14
#define      Kb5   0x24
#define      Kb6   0x44
#define      Kb7   0x12
#define      Kb8   0x22
#define      Kb9   0x42
#define      Kbo   0x11
#define      Kb0   0x21
#define      Kbd   0x41
#define      KbA   0x88
#define      KbB   0x84
#define      KbC   0x82
#define      KbD   0x81

/* combinatii '*' si o alta tasta */
#define      Kbo1  0x19
#define      Kbo2  0x39
#define      Kbo3  0x59
#define      Kbo4  0x45
#define      Kbo5  0x35
#define      Kbo6  0x55
#define      Kbo7  0x13
#define      Kbo8  0x33
#define      Kbo9  0x53
#define      Kbo0  0x31
#define      Kbod  0x51

/* combinatii '#' si o alta tasta */
#define      Kbd1  0x59
#define      Kbd2  0x69
#define      Kbd3  0x49
#define      Kbd4  0x55
#define      Kbd5  0x65
#define      Kbd6  0x45
#define      Kbd7  0x53
#define      Kbd8  0x63
#define      Kbd9  0x43
#define      Kbd0  0x61
#define      Kbdo  0x51

// coduri de eroare
#define NKb          0
#define     GI_OK       -6  /* s-a apăsat ENTER(#) după ce s-a scris ceva nou */
#define     GI_ESC      -1  /* s-a apăsat ESC(*) fără nici un număr introdus */
#define     GI_WAIT     -2  /* sa introdus un număr şi se aşteaptă ENTER sau ESC */
#define     GI_NOINPUT  -3  /* nu s-a întâmplat nimic */
#define     GI_KBRDERR  -4  /* tastă necunoscută */
#define     GI_NODATA   -5  /* s-a apăsat ENTER(#) fără nici un număr introdus */

#endif // _KBRDCODE_H_

/*************************************************************************\
**  Titlu:        KBRD.H                                                 **
**  Descriere:    Prototipurile funcţiilor pentru citirea tastaturii 3x4  **
**                sau 4x4                                                **
**                                                                       **
**  Acest modul conţine următoarele funcţii:                             **
**    Citire bufferată a tastaturii:                                     **
**       fKBRD_ERR:  flag care indică o eroare apăruta la citirea tast.   **
**       iKBRD_read(void) citeşte de la port şi depune în stivă.         **
**    Funcţii de citire a tastaturii:                                    **
**       KBHIT: flag care indică apăsarea unei taste.                    **
**       byte KBH_init(void) funcţie care iniţializează tastatura şi date **
**       char getnomber(byte row, byte col, char *txt, char retlen,      **
**            char** retsir) funcţie de editare şir de caractere         **
**                                                                       **
**  Observaţii: Pentru validarea lucrului bufferat cu tastatura trebuie   **
```

```
**              introdusa linia '#define KBRD_BUFFER'                        **
**                                                                           **
\***************************************************************************/
#ifndef      _KBRD_H_
#define _KBRD_H_

#include <typedef.h>
#include "..\main\system.h"

//#define _KBRD_BUFFER_

#ifdef _KBRD_BUFFER_
  extern bit fKBRD_ERR;           // a apărut o eroare la citirea tastaturii
  extern byte iKBRD_read(void);   // citeşte direct portul de tastatura şi depune
                                  //datele în bufferul de tastatură

#endif

extern bit KBHIT;

extern void KBRD_init(void);      // iniţializează tastatura
extern byte KBRD_read(void);      // extrage un caracter din bufferul de tastatură
//extern int getint(byte row, byte col, char *txt, char len);
extern char getnomber(byte row, byte col, char *txt, char retlen, char** retsir);

#endif

/***************************************************************************\
**  Titlu:        KBRD.C                                                     **
**  Descriere:    Implementarea funcţiilor pt. folosirea tastaurii 3x4 sau**
**                4x4                                                        **
**                                                                           **
**  Acest modul conţine următoarele funcţii:                                 **
**    Citire bufferată a tastaturii:                                         **
**       fKBRD_ERR:  flag care indica o eroare apărută la citirea tast.   **
**       iKBRD_read(void) citeşte de la port şi depune în stivă.          **
**    Funcţii de citire a tastaturii:                                        **
**       KBHIT: flag care indică apăsarea unei taste.                     **
**       byte KBH_init(void) funcţie care iniţializează tastatura şi date.**
**       char getnomber(byte row, byte col, char *txt, char retlen,       **
**            char** retsir) funcţie de editare şir de caractere          **
**                                                                           **
**  Observaţii: Pentru validarea lucrului bufferat cu tastatura trebuie   **
**              introdusa linia '#define KBRD_BUFFER' in KBRD.H           **
**              Foloseşte rutine pentru afişare pe LCD (LCD.H)            **
**                                                                           **
\***************************************************************************/
#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
#include <reg552.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <typedef.h>
#include "..\main\system.h"
#include "kbrdcode.h"
#include "kbrd.h"

bit KBHIT;

#ifdef _KBRD_BUFFER_        // Dacă se foloseşte bufferarea.
  static idata      byte   KBRD_BUFF[KBRD_BUFF_SIZE]; // bufferul de tastatură
  static idata      byte   KBRD_R,KBRD_W;             // poziţia de scriere si citire
  bit  fKBRD_ERR;          // a apărut o eroare la citirea tastaturii
  byte iKBRD_read(void);   // citeşte direct portul de tastatură şi depune datele în
                           // buffer
#endif

void KBRD_init(void);             // iniţializează tastatura
byte KBRD_read(void);             // extrage un caracter din bufferul de tastatură

\*************************************************************************
Iniţializează tastatura.
  Dacă este validat modul bufferat iniţializează pointerii de citire şi scriere în
  stiva tastaturii.
  Setează portul P4 ca fiind portul la care este legată tastatura.
*************************************************************************/
void KBRD_init(void)
{
#ifdef _KBRD_BUFFER_
  KBRD_R = 0x00;
  KBRD_W = 0x00;
#endif
  P4 = 0xff;
```

```
}

\*********************************************************************
Funcţiile de citire a tastaturii:
  byte iKBRD_read(void): citire bufferată a tastaturii. Se apelează din
                         întreruperi.
  byte KBRD_read(void): citire nebufferată a tastaturii.
*********************************************************************/
#ifdef _KBRD_BUFFER_
  byte iKBRD_read(void)
#else
  byte KBRD_read(void)
#endif
{
  data byte t,poz;
  static data byte to;

  P4 = 0xf0;          // se scot pe coloane 1111
  t = P4 | 0x0f;      // se citesc rândurile
  P4 = t;             // se scot pe rânduri
  t = ~P4;            // se citesc şi se complementează

  if(!t)
    to = 0;

  if((t == to) || (!(t & 0xf0)))
    return 0;
  to = t;

  KBHIT = 1;          // s-a apăsat o tasta

#ifdef _KBRD_BUFFER_
  // scriem în bufferul de tastatură
  KBRD_BUFF[KBRD_W] = t;
  // incrementăm poziţia de scriere în buffer numai dacă nu suprascriem
  // informaţia mai veche
  poz = (KBRD_W + 1) % KBRD_BUFF_SIZE;

  if(poz != KBRD_R)
    KBRD_W = poz;
  else
    fKBRD_ERR = 1;
#endif
  return(t);
}

\*********************************************************************
Funcţiile de citire a tastaturii:
byte KBRD_read(void): citire bufferată a tastaturii. Se apelează
atunci când se lucrează cu iKBRD_read în întreruperi.
*********************************************************************/
#ifdef _KBRD_BUFFER_
byte KBRD_read(void)
{
   data byte r;

   if(KBRD_R == KBRD_W)
     return 0;             // bufferul de tastatură este gol
   else
   {
     r = KBRD_BUFF[KBRD_R];// returnăm ce găsim în bufferul de tastatură
     KBRD_R = (KBRD_R + 1) % KBRD_BUFF_SIZE;
     return r;             // am extras un caracter din buffer
   }
}
#endif
```

2.4.2 Rutine pentru utilizarea tastaturii şi display-ului LCD în sistemele cu 80C167

Performanţele ridicate ale circuitului 80C167 permit utilizarea facilă nu numai a unui simplu display LCD alfanumeric, ci chiar şi a unui display grafic LCD şi chiar a unei tastaturi matriciale de 16 rânduri x 16 coloane (256 taste).

În scopul păstrării compatibilităţii cu perifericele sistemului de dezvoltare 80C552, aplicaţia curentă va prezenta utilizarea unui display LCD 4 rânduri x

20 caractere (LM16x21A) şi o tastatură 4x4. Folosind principiile expuse, utilizatorul poate extinde sau reduce dispozitivele la necesităţile dorite.

Modul de conectare a celor două periferice la sistemul de dezvoltare este prezentat în figura 3.5.

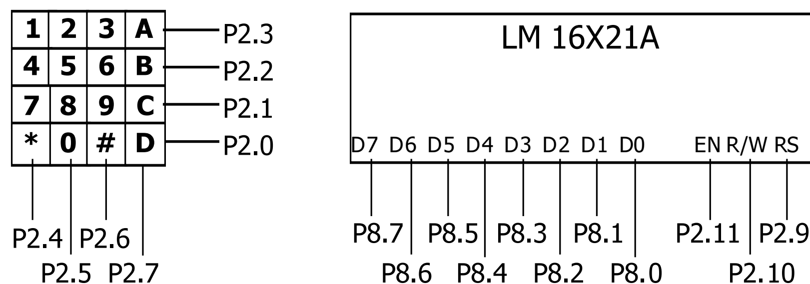


Figura 2.6. Display şi tastatură pentru 80C167

Principiul de lucru cu tastatura şi display-ul este identic cu cel descris anterior. Pentru conectarea display-ului s-a preferat comanda directă pe port I/O în locul conectării pe magistrala de date a microcontrolerului datorită vitezei foarte mici a dispozitivului şi incapacităţii circuitului 80C167 de a produce semnale suficient de lente pentru comanda acestuia.

```
/**********************************************************************\
**  Titlu:        KBRD.C                                              **
**  Descriere:    Modul pentru accesare display LM16x21A şi tastatură 4x4 **
**                                                                    **
**  Versiune:     2.0                                                 **
**  Început la:   August 97                                           **
**  Autor:        Ştefan Suceveanu, Pratco s.r.l. Bucureşti, România  **
**  Compilator C: C166, Franklin Software, Inc.                       **
**                                                                    **
**  Acest modul conţine următoarele funcţii:                          **
**      void cmdLCD(char c):       comandă LCD;                       **
**      void initLCD(void):        resetare şi iniţializare LCD;      **
**      char readLCD(void):        citeşte caracter din CG/DD RAM LCD;  **
**      void dataLCD(char c):      scrie caracter in CG/DD RAM LCD;     **
**      void putLCD(char rind, char coloana, char *s):                **
**                                 pune ASCIIZ la rând, coloană;      **
**      void chrLCD(char rind, char coloana, char c):                 **
**                                 pune char la rând, coloană;        **
**      void clrLCD(char rind):    şterge rândul specificat (4 şterge tot)**
**      unsigned int kbd(void):    citeşte un caracter de la tastatura 4*4**
**                                                                    **
**  Ultima modificare la data: 24 iunie 1998                          **
**  Istoric:                                                          **
**                                                                    **
\**********************************************************************/
#pragma MOD167

#include <reg167.h>              /* special function register 80C167      */
#include <stdio.h>               /* standard I/O .h-file                  */
#include <ctype.h>               /* standard I/O .h-file                  */
#include <intrins.h>

char waitLCD(void);              /* aşteaptă LCD ready, întoarce AC       */
void cmdLCD(char c);             /* comanda către LCD                     */
void initLCD(void);              /* reset şi iniţializare LCD             */
char readLCD(void);              /* citeşte caracter din CG/DD RAM        */
void dataLCD(char c);            /* scrie caracter in CG/DD RAM           */
void putLCD(char rind, char coloana, char *s);/*pune ASCIIZ la rând, coloană*/
void chrLCD(char rind, char coloana, char c); /*pune char la rând, coloană*/
void clrLCD(char rind);      /* sterge rindul specificat (4 sterge tot)      */
unsigned int kbd(void);      /* citeste un caracter de la Tastatura 4*4      */

char waitLCD(void)
{
  unsigned char p;

  DP2 |= 0x0e00;             /* set port 2 out (LCDcmd)        */
  DP8 = 0x00;                /* set port 8 in (date)           */

  P2 &= 0xf7ff;              /* Dezactivare LCD  (P2.11)       */
  P2 |= 0x0400;              /* R/W = 1  (P2.10)               */
  P2 &= 0xfdff;              /* RS = 0   (P2.9)                */
```

```
  P2 |= 0x0800;            /* Validare LCD                  */

  p = P8;
  while((p & 0x80) == 0x80)
    p = P8;

  P2 &= 0xf7ff;            /* Validare LCD                  */
  return(p & 0x7f);
}

void cmdLCD(char c)
{
  waitLCD();

  DP8 = 0xff;              /* set port 8 out                */
  _nop_();

  P8 = c;                  /* depune cmd la LCD             */
  P2 &= 0xf1ff;            /* dezactivare LCD,R/W = 0,RS = 0 */

  P2 |= 0x0800;            /* validare LCD                  */
  P2 &= 0xf7ff;            /* dezactivare LCD               */
}

char readLCD(void)
{
  char p;

  waitLCD();
  P2 |= 0x0e00;            /* validare LCD, R/W=1, RS=1     */
  p = P8;
  P2 &= 0xf7ff;            /* dezactivare LCD               */
  return(p);
}

void dataLCD(char c)
{
  waitLCD();

  DP8 = 0xff;              /* set port 8 out                */
  _nop_();

  P8 = c;                  /* depune data la LCD            */

  P2 |= 0x0200;            /* RS = 1                        */
  P2 &= 0xfbff;            /* R/W = 0                       */

  P2 |= 0x0800;            /* validare LCD                  */
  P2 &= 0xf7ff;            /* dezactivare LCD               */
}

void initLCD(void)
{
  cmdLCD(0x38);     /* function set.8 biţi, 2 linii, 5x7 dot matrix   */
  cmdLCD(0x06);     /* entry mode set.Increment adr, no display shift */
  cmdLCD(0x0c);     /* display ON. Display on/off control             */
  cmdLCD(0x01);     /* clear display.                                 */
}

void putLCD(char rind, char coloana, char *s)
{
unsigned char cmd;

  switch(rind)
    {
     case 0: cmd = 0x80;break;
     case 1: cmd = 0xc0;break;
     case 2: cmd = 0x94;break;
     case 3: cmd = 0xd4;break;
     default: break;
   }

  cmd += coloana;
  cmdLCD(cmd);

  for(cmd = 0; s[cmd]; cmd++)
     dataLCD(s[cmd]);
}

void chrLCD(char rind, char coloana, char c)
{
unsigned char cmd;
```

```
  switch(rind)
    {
     case 0: cmd = 0x80;break;
     case 1: cmd = 0xc0;break;
     case 2: cmd = 0x94;break;
     case 3: cmd = 0xd4;break;
     default: break;
   }

  cmd += coloana;
  cmdLCD(cmd);
  dataLCD(c);
}

void clrLCD(char rind)
{
unsigned char cmd;
bit sters = 1;

  switch(rind)
    {
     case 0: cmd = 0x80;break;
     case 1: cmd = 0xc0;break;
     case 2: cmd = 0x94;break;
     case 3: cmd = 0xd4;break;
     default: cmd = 0x01;sters=0;break;
    }

  cmdLCD(cmd);
  if(sters)
       for(cmd=0; cmd< 20;cmd++)
              dataLCD(' ');
}

unsigned int kbd(void)      /* Tastatura 4*4                  */
{
  unsigned int t;
  static unsigned int to;

  DP2 = 0x000f;             /* P2.0-3 set OUT                 */
  ODP2 = 0x00ff;            /* P2.0-7 set OPEN DRAIN          */
  _nop_();
  P2 = 0;                   /* OUT                            */

  DP2 = 0x0e0f;             /* P2.4-7 set IN                  */
  _nop_();                  /* P2.9-B pentru LCD              */
  t = P2 & 0x00f0;          /* IN                             */

  DP2 = 0x0ef0;             /* P2.4-7 set OUT                 */
                            /* P2.0-3 set IN                  */
  _nop_();
  P2 = t;                   /* OUT value                      */

  t |= (P2 & 0x000f);       /* IN or value                    */

  t = (~t) & 0x007f;

  if(!t)
    to = 0;

  if((t == to) || (!(t & 0x000f)))
    return 0;
  to = t;

  switch (t)                /* conversie ASCII                */
    {
    case 0x11: t=0x30;break;      /* ‘0’ */
    case 0x21: t=0x31;break;      /* ’1’ */
    case 0x41: t=0x32;break;      /* ‘2’ */
    case 0x01: t=0x33;break;      /* ‘3’ */
    case 0x12: t=0x34;break;      /* ‘4’ */
    case 0x22: t=0x35;break;      /* ‘5’ */
    case 0x42: t=0x36;break;      /* ‘6’ */
    case 0x02: t=0x37;break;      /* ‘7’ */
    case 0x14: t=0x38;break;      /* ‘8’ */
    case 0x24: t=0x39;break;      /* ‘9’ */
    case 0x44: t=0x41;break;      /* ‘A’ */
    case 0x04: t=0x42;break;      /* ‘B’ */
    case 0x18: t=0x43;break;      /* ‘C’ */
    case 0x28: t=0x44;break;      /* ‘D’ */
    case 0x48: t=0x45;break;      /* ‘E’ */
    case 0x08: t=0x46;break;      /* ‘F’ */
    default:   t=0;   break;
```

```
    }
  return(t);
}
```

2.4.3 Funcţii de citire şi editare şiruri de caractere

Tot legat de citirea tastaturii mai este prezentată şi o funcţie mai complexă care foloseşte şi modulul de afişare pe LCD (`getnumber`). Această funcţie este folosită pentru afişarea unui mesaj şi citirea unei variabile numerice care este returnată sub forma unui şir de caractere de lungime predefinită la apelul funcţiei. În această funcţie au fost atribuite următoarele definiţii tastelor:

A: introduce un punct zecimal (dacă nu a fost introdus deja unul)
B: introduce semnul – pe prima poziţie.
C: şterge ultimul caracter
*: ESC
#: ENTER

Funcţia nu aşteaptă ca să fie apăsata o tastă şi deci trebuie apelată în polling.

Funcţia `EDITTEXT` pentru editare de texte implementată este un mic editor de texte. Aceasta primeşte ca parametrii un mesaj, `MSG`, care va fi afişat pe primul rând pe afişajul LCD, un text, `TXT`, care va fi afişat pe rândul al doilea pe LCD şi care reprezintă valoarea implicită care va fi modificată de utilizator. Următoarele interpretări s-au dat tastelor:

TASTA 4: mută cursorul la stânga; scroll text dacă este necesar;
TASTA 6: mută cursorul la dreapta; scroll text dacă este necesar;
TASTA 2: incrementează caracterul de sub cursor;
TASTA 2: decrementează caracterul de sub cursor;
TASTA 5: caracterul de sub cursor = spaţiu;
TASTA 1: caracterul de sub cursor = 'A';
TASTA 3: caracterul de sub cursor = 'Z';
TASTA 7: caracterul de sub cursor = 'a';
TASTA 9: caracterul de sub cursor = 'z';
TASTA 0: caracterul de sub cursor = '0';
TASTA *: ESC funcţia va întoarce GI_ESC
TASTA #: ENTER funcţia va întoarce 1

Funcţia nu aşteaptă ca să fie apăsata o tastă şi deci trebuie apelată în polling.

```
/*********************************************************************\
**  Titlu:         EDITTEXT.C                                          **
**  Descriere:     Citirea unui text si a unui număr                   **
**                                                                     **
**  Versiune:      2.0                                                 **
**  Început la:    August 97                                           **
**  Autor:         Ştefan Suceveanu, Pratco s.r.l. Bucureşti, România  **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                   **
**                                                                     **
**  Acest modul conţine următoarele funcţii:                           **
**       int edittext(char* msg, char* txt, char maxlen)               **
**           Afişează msg pe LCD pe primul rând şi txt pe rândul 2     **
**           Modifica txt, care poate să conţină maxim maxlen caractere  **
**       char getnomber(byte row, byte col, char *txt, char len,       **
```

```
**                      char** retsir)                                      **
**            Scrie txt pe rândul şi coloana [row,col] pe LCD                **
**            Citeşte maxim len digiţi care sunt returnaţi în retsir        **
**                                                                          **
**   Ultima modificare la data: 24 iunie 1998                               **
**   Istoric:                                                               **
**                                                                          **
**   Observaţii:                                                            **
**                                                                          **
\***************************************************************************/
#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
#include <reg552.h>
#include <typedef.h>
#include <string.h>
#include "kbrd.h"           // funcţii pentru citirea tastaturii
#include "kbrdcode.h"       // codurile returnate de tastatură
#include "lcd.h"            // funcţii pentru lucru cu LCD

extern xdata int kbstate,   // starea citirii tastaturii
                kbsstate;   // substarea citirii tastaturii

#define Nsir  5             // int are maximum 5 cifre -32768 32767
xdata char l[Nsir+1];       // buffer pentru numărul citit de la tastatura

/* Editează un text
        afişează MSG pe primul rand
        afişează txt pe rândul 2 si îl modifică
        se pot introduce maxim 'maxlen' caractere
        txt trebuie alocat txt[maxlen+1]
TASTA 4:        mută cursorul la stânga; scroll text dacă este necesar
TASTA 6:        mută cursorul la dreapta; scroll text dacă este necesar
TASTA 2:        incrementează caracterul de sub cursor
TASTA 2:        decrementează caracterul de sub cursor
TASTA 5:        caracterul de sub cursor = spaţiu
TASTA 1:        caracterul de sub cursor = 'A'
TASTA 3:        caracterul de sub cursor = 'Z'
TASTA 7:        caracterul de sub cursor = 'a'
TASTA 9:        caracterul de sub cursor = 'z'
TASTA 0:        caracterul de sub cursor = '0'

INTOARCE:       GI_WAIT = se aşteaptă ENTER(#) sau ESC(*)
                GI_ESC  = s-a apăsat ESC(*)
                1       = s-a apăsat ENTER(#)
!ATENTIE!       modifica şi foloseşte variabila globală kbstate
                pentru iniţializare kbsstate trebuie setată cu 0
!ATENTIE!       modifică şi foloseşte variabila globală kbstate
                La ieşire kbstate se iniţializează cu 0
*/
int edittext(char* msg, char* txt, char maxlen)
{
  static data char t,poz,s;
  data byte c;

  if(!kbsstate)
  {
    poz = s = 0;
    LCDstr(0, 0, msg);
    LCDstr(1, 0, txt);
    LCDstr(1, 0, "\0");
    kbsstate++;
    CursorON();
  }
  else
  {
    t = KBRD_read();
    if(!t)
      return GI_WAIT;
    switch(t)
    {
      case Kb4: if(poz)                  //deplasează cursor la stânga, scroll
                  poz--;
                if(poz < s)
                {
                  s = poz;
                  LCDclr(1);
                  LCDstr(1, 0, &txt[s]);
                  LCDstr(1, 0, "\0");
                }
                else
                  CursorLeft();
                break;
      case Kb6: poz++;                   //deplasează cursor la dreapta, scroll
                if(poz > strlen(txt))
```

```
                poz = strlen(txt);
              if(poz > maxlen)
                poz = maxlen;
              if(poz > s+15)
              {
                s = poz - 15;
                LCDclr(1);
                LCDstr(1, 0, &txt[s]);
                LCDstr(1, 15, "\0");
              }
              else
                LCDstr(1, poz-s, "\0");
//              CursorRight();
              break;
      case Kb2: if(poz == strlen(txt))   //incrementează caracterul de sub cursor
              {
                txt[poz+1] = 0;        //ultimul caracter -> ‘0’
                txt[poz] = '0';
              }
              else
              {
                c = txt[poz];
                c++;
                if(c > 0x7f)
                  c = 0x20;
                txt[poz] = c;
              }
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb8: if(poz == strlen(txt))   //decrementează caracterul de sub cursor
              {
                txt[poz+1] = 0;        //ultimul caracter -> ‘z’
                txt[poz] = 'z';
              }
              else
              {
                c = txt[poz];
                c--;
                if(c < 0x20)
                  c = 0x7f;
                txt[poz] = c;
              }
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb5: txt[poz] = 0x20;         //caracterul de sub cursor = ‘ ’
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb0: txt[poz] = '0';          //caracterul de sub cursor = ‘0’
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb1: txt[poz] = 'A';          //caracterul de sub cursor = ‘A’
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb3: txt[poz] = 'Z';          //caracterul de sub cursor = ‘Z’
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb7: txt[poz] = 'a';          //caracterul de sub cursor = ‘a’
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case Kb9: txt[poz] = 'z';          //caracterul de sub cursor = ‘z’
              LCDstr(1, 0, &txt[s]);
              LCDstr(1, poz, "\0");
              break;
      case KbC: strcpy(&txt[poz], &txt[poz+1]); //şterge caracterul
              LCDstr(1, 0, &txt[s]);
              break;
      case Kbo: kbstate = 0;                    //‘*’ = ESC
              LCDclr(0);
              LCDclr(1);
              CursorOFF();
              return GI_ESC;
              break;
      case Kbd: kbstate = 0;                    //‘#’ = ENTER
              LCDclr(0);
              LCDclr(1);
              CursorOFF();
```

```
                return 1;
                break;
      default:
                break;
    }
  }
  return GI_WAIT;
}

/* **********************************************************************
Returnează în 'retsir' adresa către textul care s-a introdus (numai cifre)
Returnează un char cu următoarea semnificaţie
GI_ESC        -1     s-a apăsat ESC(*) fără nici un număr introdus
GI_WAIT       -2     s-a introdus un număr şi se aşteaptă ENTER sau ESC
GI_NOINPUT    -3     nu s-a întâmplat nimic
GI_KBRERR     -4     tastă necunoscută
GI_NODATA     -5     s-a apăsat ENTER(#) fără nici un număr introdus
GI_OK         -6     s-a apăsat ENTER(#) după ce s-a introdus ceva

Parametrii la apel:
   row,col:     rândul si coloana pe care se va scrie mesajul
   txt:         un mesaj care se va afişa pe LCD
   len:         numărul de caractere citite
   retsir:      pointer la şirul de caractere introdus
*********************************************************************** */
char getnomber(byte row, byte col, char *txt, char len, char** retsir)
{
  static bit first=1,refresh=0,fput=1,virgula;
  static char k=0;

  *retsir = NULL;    // numai daca GI_OK se întoarce pointer la şir

  if(len < 0) // reset
  {
    first = 1;
    fput = 1;
    refresh = 0;
    k = 0;
    return GI_NOINPUT;
  }

  if(fput)
  {
    BlinkON();
    if(txt)
      LCDstr(row, col, txt);
    else
      LCDstr(row, col, "\0");
    fput=0;
  }

  if(!k)
    virgula = 0;

  switch(KBRD_read())
    {
     case Kb0:
              if(k < len && k < Nsir)
                l[k++] = '0';
              first = 0;
              refresh = 1;
                break;

     case Kb1:
              if(k < len && k < Nsir)
                l[k++] = '1';
              first = 0;
              refresh = 1;
                break;

     case Kb2:
              if(k < len && k < Nsir)
                l[k++] = '2';
              first = 0;
              refresh = 1;
                break;

     case Kb3:
              if(k < len && k < Nsir)
                l[k++] = '3';
              first = 0;
              refresh = 1;
                break;
```

```
case Kb4:
        if(k < len && k < Nsir)
          l[k++] = '4';
        first = 0;
        refresh = 1;
          break;

case Kb5:
        if(k < len && k < Nsir)
          l[k++] = '5';
        first = 0;
        refresh = 1;
          break;

case Kb6:
        if(k<len && k < Nsir)
          l[k++] = '6';
        first = 0;
        refresh = 1;
          break;

case Kb7:
        if(k<len && k < Nsir)
          l[k++] = '7';
        first = 0;
        refresh = 1;
          break;

case Kb8:
        if(k<len && k < Nsir)
          l[k++] = '8';
        first = 0;
        refresh = 1;
          break;

case Kb9:
        if(k<len && k < Nsir)
          l[k++] = '9';
        first = 0;
        refresh = 1;
          break;

case KbA:                   // punct zecimal
        if(!virgula)        // dacă nu avem nici o virgula
        {
          if(k<len && k < Nsir)
            l[k++] = '.';   // punem o virgulă
          first = 0;
          refresh = 1;
          virgula = 1;
        }
          break;

case KbB:      if(first)    // semn doar în prima poziţie
          {
          l[k++] = '-';
          first = 0;
          refresh = 1;
          }
          break;

case KbC:      if(!first)   // ştergem ultimul caracter
          {
          k--;
          if(l[k] == '.')   // dacă ştergem virgula
            virgula = 0;    // marcăm că nu avem virgulă
          l[k] = ' ';
          if(txt)
            LCDstr(row, col + strlen(txt), l);
          else
            LCDstr(row, col, l);
          l[k] = 0;
          if(!k)
            first = 1;
          refresh = 1;
          }
          break;

case Kbo:  // c = '*';      // ESC
        k = 0;
        first = 1;
        fput = 1;
```

```
              refresh = 0;
             LCDcmd(0x0c);
               return GI_ESC;    // nici un număr introdus
             break;

    case Kbd:  // c = '#';              // ENTER
             if(first)
             {
               k = 0;
               first = 1;
               fput = 1;
                refresh = 0;
               BlinkOFF();
               return GI_NODATA;        // nici un număr introdus
             }
               else
             {
               k = 0;
               first = 1;
               fput = 1;
                refresh = 0;
               BlinkOFF();
               *retsir = l;
                      return GI_OK;      // avem ceva in retcode
             }
               break;

//     case NKb:
//           if(first)                  // nu este apăsată nici o tastă
//             return GI_NOINPUT;
//           break;

//     default:   return GI_KBRDERR;     // tastă necunoscută
//           break;
    }

  if(refresh)
  {
    refresh = 0;
    l[k] = 0;
    if(txt)
    {
      LCDstr(row, col, txt);
      LCDstr(row, col + strlen(txt), l);
    }
    else
      LCDstr(row, col, l);
  }

  if(first)         // nu este apăsată nici o tastă
    return GI_NOINPUT;

  return GI_WAIT;
}
```

2.5. Extinderea capacităţilor aritmetice

Realizarea unor aplicaţii rapide în asamblor poate fi dificilă în situaţia în care algoritmul trebuie să prelucreze valori numerice. O problemă poate consta în conversia din cod hexazecimal, folosit intern pentru reprezentarea numerelor, în cod zecimal, necesar pentru o afişare inteligibilă pe un display sau un terminal. O altă chestiune poate fi creşterea preciziei de reprezentare a numerelor, de exemplu de la 8 biţi (numere pozitive în domeniul 0...255) la 32 de biţi (numere pozitive în domeniul 0...4 294 967 295), sau reprezentarea şi operaţiile aritmetice cu numere raţionale.

2.5.1 Aritmetică BCD

Pentru circuitul 80C552 realizarea unor calcule în cod BCD este facilitată de existenţa instrucţiunii DA (ajustare zecimală a acumulatorului pentru adunare). Circuitul 80C167 nu are o astfel de instrucţiune şi rutina nu poate fi adaptată pentru acesta.

Pentru creşterea vitezei de execuţie se recomandă efectuarea completă a calculelor în hexazecimal, urmând ca numai datele care urmează să fie afişate (pe terminal sau pe un display LCD) să fie convertite în cod BCD. Procedura este recomandată să fie făcută în asamblor, evitând funcţia C `sprintf`.

Programul respectiv foloseşte pentru conversie un tabel unde sunt memorate valorile BCD a fiecărui semioctet, funcţie de poziţia acestuia.

```
/***********************************************************************\
**  Descriere:    Rutină conversie hexazecimal BCD                     **
**                                                                     **
**  Versiune:     1.0                                                  **
**  Inceut la:    Iunie 96                                            **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania   **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                   **
**                                                                     **
**  Acest modul contine urmatoarele functii:                           **
**                                                                     **
**  Ultima modificare la data: 23 nov 1998                             **
**  Istoric:                                                           **
**                                                                     **
**  Observatii:                                                        **
**      hex0-1 octeţii cod hexazecimal                                 **
**      char5-0 rezultat sute de mii – unităţi                         **
**                                                                     **
\***********************************************************************/
hex0      data      53h             ;numarul care urmeaza sa fie hex1  data   54h
          ;convertit
char0     data      33h             ;rezultatul conversiei char1       data   34h
          ; max "65535"char2        data    35h
char3     data      36h
char4     data      37h
char5     data      38h             ;'\0'

r_0       data      55h             ; regsitrii de lucru
r_1       data      56h
r_2       data      57h


h_b:      clr       a               ;iniţializare registre lucru
          mov       r_0,a
          mov       r_1,a
          mov       r_2,a

          mov       a,hex0          ;formare semioctet 1
          anl       a,#0fh
          jz        etc3
          mov       dptr,#tab_00-1  ;conversie unităţi
          movc      a,@a+dptr
          mov       r_0,a

etc3:     mov       a,hex0
          anl       a,#0f0h
          jz        etc4
          swap      a               ;formare semioctet 2
          mov       r7,a
          mov       dptr,#tab_10-1  ;conversie zeci
          movc      a,@a+dptr
          add       a,r_0
          da        a               ;formare z+u
          mov       r_0,a
          mov       a,r7            ;valoarea unui octet este
          mov       dptr,#tab_11-1  ;zecimal intre 0 si 256
          movc      a,@a+dptr       ;acum determinam sutele
          addc      a,r_1           ;adunam cu carry
          da        a
          mov       r_1,a

etc4:     mov       a,hex1          ;formare semioctet 3
          anl       a,#0fh
          jz        etc5
          mov       r7,a
          mov       dptr,#tab_20-1  ;conversie zeci, unitati
          movc      a,@a+dptr
          add       a,r_0
          da        a               ;formare z+u
          mov       r_0,a
          mov       a,r7
          mov       dptr,#tab_21-1  ;conversie mii, sute
```

```
          movc      a,@a+dptr
          addc      a,r_1
          da        a               ;formare m+s
          mov       r_1,a

etc5:     mov       a,hex1          ;formare semioctet 4
          anl       a,#0f0h
          jz        etc6
          swap      a
          mov       r7,a
          mov       dptr,#tab_30-1  ;conversie zeci, unitati    movc   a,@a+dptr
          add       a,r_0
          da        a               ;formare z+u
          mov       r_0,a
          mov       a,r7
          mov       dptr,#tab_31-1  ;conversie mii, sute
          movc      a,@a+dptr
          addc      a,r_1
          da        a               ;formare m+s
          mov       r_1,a
          mov       a,r7
          mov       dptr,#tab_32-1  ;conversie zm
          movc      a,@a+dptr
          addc      a,r_2
          da        a               ;formatare zm
          mov       r_2,a

etc6:                               ;conversie ASCII
          mov       r2,#3
          mov       r1,r_2
          mov       r0,#char0

etc7:     mov       a,@r1
          and       a,#0f0h
          swap      a
          add       a,#40h
mov       @r0,a
          inc       r0
          mov       a,@r1
          and       a,#0fh
          add       a,#40h
          mov       @r0,a
          inc       r0
          dec       r1
          dec       r2
          jpnz      etc7
          mov       @r0,#00h
          ret

tab_00:   db        1,2,3,4,5,6,7,8,9
          db        10h,11h,12h,13h,14h,15h
tab_10:   db        16h,32h,48h,64h,80h,96h,12h,28h
          db        44h,60h,76h,92h,08h,24h,40h
tab_11:   db        0,0,0,0,0,0,1,1,1,1,1,1,2,2,2
tab_20:   db        56h,12h,68h,24h,80h,36h,92h,48h
          db        04h,60h,16h,72h,28h,84h,40h
tab_21:   db        2,5,7,10h,12h,15h,17h,20h
          db        23h,25h,28h,30h,33h,35h,38h
tab_30:   db        96h,92h,88h,84h,80h,76h,72h,68h
          db        64h,60h,56h,52h,48h,44h,40h
tab_31:   db        40h,81h,22h,63h,4,45h,86h,27h
          db        68h,9,50h,91h,32h,73h,14h
tab_32:   db        0,0,1,1,2,2,2,3,3,4,4,4,5,5,6
```

2.5.2 Creşterea preciziei de reprezentare a numerelor în virgulă fixă şi virgulă flotantă

Unele aplicaţii realizate în asamblor pot necesita precizii aritmetice mai mari decât cele implicite pe 8 biţi (în situaţia circuitului 80C552).

În cele ce urmează sunt prezentate câteva rutine utile pentru lucrul pe 16 de biţi. Extinderea dimensiunii datelor de la 8 la 16 de biţi asigură creşterea posibilităţilor de reprezentare a numerelor întregi, de la [0-255] la [0-4294967285].

Pentru funcţia transcedentală introdusă (sinus) asigurarea unei precizii de 16 de biţi este dificil de realizat: implementarea unei dezvoltări în serie

depăşeşte clar posibilităţile unor microcontrolere cu posibilităţi aritmetice reduse (de exemplu 80C51) iar determinarea valorilor pe bază de tabel cu valori memorate este prohibitivă datorită dimensiunilor foarte mari a acestuia. Rutinele care urmează determină funcţia sinus folosind un tabel de 256 de valori, creşterea preciziei la 16 biţi fiind asigurată de un algoritm de interpolare.

```
/*********************************************************************\
**  Descriere:    Rutină afişare vectori pe osciloscop               **
**                                                                   **
**  Versiune:     1.0                                                **
**  Inceut la:    Iunie 94                                           **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                 **
**                                                                   **
**  Acest modul contine urmatoarele functii:                         **
**          load_16                                                  **
**          load_32                                                  **
**          mul_16                                                   **
**          div_16                                                   **
**          add_16                                                   **
**          sub_16                                                   **
**          add_32                                                   **
**          sub_32                                                   **
**          low_16,mid_16,high_16                                    **
**          sinus                                                    **
**                                                                   **
**  Ultima modificare la data: 23 nov 1998                           **
**  Istoric:                                                         **
**                                                                   **
**  Observatii:                                                      **
**                                                                   **
\*********************************************************************/
PUBLIC  load_16, ?load_16?byte
PUBLIC  load_32, ?load_32?byte
PUBLIC  mul_16, ?mul_16?byte
PUBLIC  div_16, ?div_16?byte
PUBLIC  add_16, ?add_16?byte
PUBLIC  sub_16, ?sub_16?byte
PUBLIC  add_32, ?add_32?byte
PUBLIC  sub_32, ?sub_32?byte
PUBLIC  low_16, mid_16, high_16
PUBLIC  sinus,?sinus?byte

math_32_data    SEGMENT DATA
math_32_code    SEGMENT CODE
sinus_DATA      SEGMENT DATA
sinus_CODE      SEGMENT CODE

RSEG    math_32_data
?load_16?byte:  DS 2
?load_32?byte:  DS 4
?mul_16?byte:   DS 2
?div_16?byte:   DS 2
?add_16?byte:   DS 2
?sub_16?byte:   DS 2
?add_32?byte:   DS 4
?sub_32?byte:   DS 4
op_0:           DS 1
op_1:           DS 1
op_2:           DS 1
op_3:           DS 1
tmp_0:          DS 1
tmp_1:          DS 1
tmp_2:          DS 1
tmp_3:          DS 1

RSEG    math_32_code
load_16:
        ;Încarcă octeţii 0+1 ai operandului cu valoarea dorită
        mov     op_3,#0
        mov     op_2,#0
        mov     op_1,?load_16?byte
        mov     op_0,?load_16?byte + 1
        ret

load_32:
        ; Încarcă octeţii 0-4 ai operandului cu valoarea dorită
```

```
        mov     op_3,?load_32?byte
        mov     op_2,?load_32?byte + 1
        mov     op_1,?load_32?byte + 2
        mov     op_0,?load_32?byte + 3
        ret

low_16:
        ;întoarce octeţii 0+1 (LSB) ai operandului
        mov     r6,op_1
        mov     r7,op_0
        ret

mid_16:
        ; întoarce octeţii 1+2 ai operandului
        mov     r6,op_2
        mov     r7,op_1
        ret

high_16:
        ; întoarce octeţii 2+3 (MSB) ai operandului
        mov     r6,op_3
        mov     r7,op_2
        ret

add_16:
        ;adaugă doi octeţi furnizaţi de program la operand
        clr     c
        mov     a,op_0
        addc    a,?add_16?byte + 1      ;octet LSB
        mov     op_0,a
        mov     a,op_1
        addc    a,?add_16?byte          ;octet MSB + C
        mov     op_1,a
        mov     a,op_2
        addc    a,#0                    ;propagare C
        mov     op_2,a
        mov     a,op_3
        addc    a,#0                    ;propagare carry
        mov     op_3,a
        ret

add_32:
        ; adaugă patru octeţi furnizaţi de program la operand
        clr     c
        mov     a,op_0
        addc    a,?add_32?byte + 3      ;octet 0 (lsb)
        mov     op_0,a
        mov     a,op_1
        addc    a,?add_32?byte + 2      ;octet 1 + carry
        mov     op_1,a
        mov     a,op_2
        addc    a,?add_32?byte + 1      ;octet 2 + carry
        mov     op_2,a
        mov     a,op_3
        addc    a,?add_32?byte          ;octet 3 (msb) + carry
        mov     op_3,a
        ret

sub_16:
        ;scădere 16 biţi furnizaţi de program apelant din operand
        clr     c
        mov     a,op_0
        subb    a,?sub_16?byte + 1      ;octet LSB
        mov     op_0,a
        mov     a,op_1
        subb    a,?sub_16?byte          ;octet MSB + C
        mov     op_1,a
        mov     a,op_2
        subb    a,#0                    ;propagare C
        mov     op_2,a
        mov     a,op_3
        subb    a,#0                    ;propagare C
        mov     op_3,a
        ret

sub_32:
        ; scădere 32 biţi furnizaţi de program apelant din operand
        clr     c
        mov     a,op_0
        subb    a,?sub_32?byte + 3      ;octet 0
        mov     op_0,a
        mov     a,op_1
        subb    a,?sub_32?byte + 2      ;octet 1 + carry
```

```
        mov     op_1,a
        mov     a,op_2
        subb    a,?sub_32?byte + 1      ;octet 2 + carry
        mov     op_2,a
        mov     a,op_3
        subb    a,?sub_32?byte          ;octet 3 + carry
        mov     op_3,a
        ret

mul_16:
        ;înmulţire operand 32 biţi cu valoare 16 biţi
        mov     tmp_3,#0        ;ştergere 16 biţi (MSB)
        mov     tmp_2,#0
        ;generare octet 0 resultat
        mov     b,op_0
        mov     a,?mul_16?byte+1
        mul     ab
        mov     tmp_0,a
        mov     tmp_1,b
        ;generare octet 1 resultat
        mov     b,op_1
        mov     a,?mul_16?byte+1
        mul     ab
        add     a,tmp_1
        mov     tmp_1,a
        mov     a,b
        addc    a,tmp_2
        mov     tmp_2,a
        jnc     mul_loop1
        inc     tmp_3
mul_loop1:
        mov     b,op_0
        mov     a,?mul_16?byte
        mul     ab
        add     a,tmp_1
        mov     tmp_1,a
        mov     a,b
        addc    a,tmp_2
        mov     tmp_2,a
        jnc     mul_loop2
        inc     tmp_3
mul_loop2:
        ;Generare octet 2 rezultat
        mov     b,op_2
        mov     a,?mul_16?byte+1
        mul     ab
        add     a,tmp_2
        mov     tmp_2,a
        mov     a,b
        addc    a,tmp_3
        mov     tmp_3,a
        ;Generare octet 3 rezultat
        mov     b,op_1
        mov     a,?mul_16?byte
        mul     ab
        add     a,tmp_2
        mov     tmp_2,a
        mov     a,b
        addc    a,tmp_3
        mov     tmp_3,a
        ;Finalizare rezultat
        mov     b,op_3
        mov     a,?mul_16?byte+1
        mul     ab
        add     a,tmp_3
        mov     tmp_3,a
        ;Eliminare rezultat superior. Valoare numai pe 32 biţi
        mov     b,op_2
        mov     a,?mul_16?byte
        mul     ab
        add     a,tmp_3
        mov     tmp_3,a
        ;Mutare din registre temporare în operand
        mov     op_0,tmp_0
        mov     op_1,tmp_1
        mov     op_2,tmp_2
        mov     op_3,tmp_3
        ret

div_16:
        ;împărţire operand 32 biţi cu valoare
        mov     r7,#0
        mov     r6,#0           ;ştergere rest
```

```
        mov     tmp_0,#0
        mov     tmp_1,#0
        mov     tmp_2,#0
        mov     tmp_3,#0
        mov     r1,?div_16?byte ;încărcare divizor
        mov     r0,?div_16?byte+1
        mov     r5,#32          ;număr bucle
        ;start împărţire
div_loop:
        call    shift_d         ;deplasare împărţitor şi întoarce MSB în C
        mov     a,r6            ;deplasare C în LSB rest
        rlc     a
        mov     r6,a
        mov     a,r7
        rlc     a
        mov     r7,a
        ;test r7:r6 >= r1:r0
        clr     c
        mov     a,r7            ;scădere r1-r7 pentru comparare r1 < r7
        subb    a,r1            ; a = r7 - r1, c setat dacă r7 < r1
        jc      cant_sub
        ; r7>r1 sau r7=r1
        jnz     can_sub         ;salt dacă r7>r1
        ;dacă r7 = r1, test dacă r6>=r0
        clr     c
        mov     a,r6
        subb    a,r0            ; a = r6 - r0, c setat dacă r6 < r0
        jc      cant_sub
can_sub:
        ;scădere divizor din rest
        clr     c
        mov     a,r6
        subb    a,r0            ; a = r6 - r0
        mov     r6,a
        mov     a,r7
        subb    a,r1            ; a = r7 - r1 - c
        mov     r7,a
        setb    c               ; deplasare 1 în cât
        jmp     quot
cant_sub:
        clr     c               ; deplasare 0 în cât
quot:   call    shift_q         ; deplasare c în cât
        djnz    r5,div_loop     ; terminat?
        ; mutare registre temporare în operand
        mov     op_0,tmp_0
        mov     op_1,tmp_1
        mov     op_2,tmp_2
        mov     op_3,tmp_3
        ret

shift_d:
        ;deplasare divizor un bit la stânga; MSB în C
        clr     c
        mov     a,op_0
        rlc     a
        mov     op_0,a
        mov     a,op_1
        rlc     a
        mov     op_1,a
        mov     a,op_2
        rlc     a
        mov     op_2,a
        mov     a,op_3
        rlc     a
        mov     op_3,a
        ret

shift_q:
        ;deplasare cât un bit la stânga; C în LSB
        mov     a,tmp_0
        rlc     a
        mov     tmp_0,a
        mov     a,tmp_1
        rlc     a
        mov     tmp_1,a
        mov     a,tmp_2
        rlc     a
        mov     tmp_2,a
        mov     a,tmp_3
        rlc     a
        mov     tmp_3,a
        ret
```

```
RSEG       sinus_DATA
?sinus?byte:
           DS 2       ; argumentul si rezultatul (HIGH, LOW)

RSEG       sinus_CODE
; Argumentul este in domeniul 0h-0ffffh caruia ii corespunde
; un domeniu 0-90 grade
; Rezultatul este in domeniul 0h-7fffh caruia ii corespunde
; un domeniu 0-1
sinus:
           mov        A,?sinus?byte      ; HIGH argument
           mov        DPTR,#sin_tab      ; inceput tabel sinus

           mov        B,#2               ; citim sin(HIGH(i))
           mul        AB
           add        A,DPL
           mov        DPL,A
           mov        A,DPH
           adc        A,B
           mov        DPH,A


           movc       A,@DPTR
           mov        R6,A               ; HIGH sin(HIGH(i))
           inc        DPTR
           movc       A,@DPTR
           mov        R7,A               ; LOW sin(HIGH(i))

           inc        DPTR
           mov        R4,A               ; HIGH sin(HIGH(i+1))
           inc        DPTR
           movc       A,@DPTR
           mov        R5,A               ; LOW sin(HIGH(i+1))

           clr        c                  ; sin(HIGH(i+1) – sin(HIGH(i))
           mov        A,R5
           subb       A,R7
           mov        A,R4
           subb       A,R6               ; panta

           mov        B,?sinus?byte + 1  ;
           mul        AB                 ; panta * LOW(i)

           add        A,R7               ; sin(HIGH(i)) + panta * LOW(i)
           mov        ?sinus?byte,A
           mov        A,B
           adc        A,R6
           mov        ?sinus?byte+1,A

           ret

sin_tab:   DW         0000h,00c9h,0192h,025bh,0324h,03edh,04b6h,057fh,
                      0647h,0710h,07d9h,08a2h,096ah,0a33h,0afbh,0bc3h,
                      0c8bh,0d53h,0e1bh,0ee3h,0fabh,1072h,1139h,1201h,
                      12c8h,138eh,1455h,151bh,15e2h,16a8h,176dh,1833h,
                      18f8h,19bdh,1a82h,1b47h,1c0bh,1ccfh,1d93h,1e56h,
                      1f19h,1fdch,209fh,2161h,2223h,22e5h,23a6h,2467h,
                      2528h,25e8h,26a8h,2767h,2826h,28e5h,29a3h,2a61h,
                      2b1fh,2bdch,2c98h,2d55h,2e11h,2ecch,2f87h,3041h,
                      30fbh,31b5h,326eh,3326h,33deh,3496h,354dh,3604h,
                      36bah,376fh,3824h,38d8h,398ch,3a40h,3af2h,3ba5h,
                      3c56h,3d07h,3db8h,3e68h,3f17h,3fc5h,4073h,4121h,
                      41ceh,427ah,4325h,43d0h,447ah,4524h,45cdh,4675h,
                      471ch,47c3h,4869h,490fh,49b4h,4a58h,4afbh,4b9eh,
                      4c3fh,4ce1h,4d81h,4e21h,4ebfh,4f5eh,4ffbh,5097h,
                      5133h,51ceh,5269h,5302h,539bh,5433h,54cah,5560h,
                      55f5h,568ah,571dh,57b0h,5842h,58d4h,5964h,59f3h,
                      5a82h,5b10h,5b9dh,5c29h,5cb4h,5d3eh,5dc7h,5e50h,
                      5ed7h,5f5eh,5fe3h,6068h,60ech,616fh,61f1h,6271h,
                      62f2h,6371h,63efh,646ch,64e8h,6563h,65ddh,6657h,
                      66cfh,6746h,67bdh,6832h,68a6h,6919h,698ch,69fdh,
                      6a6dh,6adch,6b4ah,6bb8h,6c24h,6c8fh,6cf9h,6d62h,
                      6dcah,6e30h,6e96h,6efbh,6f5fh,6fc1h,7023h,7083h,
                      70e2h,7141h,719eh,71fah,7255h,72afh,7307h,735fh,
                      73b5h,740bh,745fh,74b2h,7504h,7555h,75a5h,75f4h,
                      7641h,768eh,76d9h,7723h,776ch,77b4h,77fah,7840h,
                      7884h,78c7h,7909h,794ah,798ah,79c8h,7a05h,7a42h,
                      7a7dh,7ab6h,7aefh,7b26h,7b5dh,7b92h,7bc5h,7bf8h,
                      7c29h,7c5ah,7c89h,7cb7h,7ce3h,7d0fh,7d39h,7d62h,
                      7d8ah,7db0h,7dd6h,7dfah,7e1dh,7e3fh,7e5fh,7e7fh,
                      7e9dh,7ebah,7ed5h,7ef0h,7f09h,7f21h,7f38h,7f4dh,
                      7f62h,7f75h,7f87h,7f97h,7fa7h,7fb5h,7fc2h,7fceh,
                      7fd8h,7fe1h,7fe9h,7ff0h,7ff6h,7ffah,7ffdh,7fffh,
```

7fffh

Pentru calcule în virgulă flotantă, utilizatorul trebuie să-şi stabilească mai întâi formatul de reprezentare al acestor numere. Un număr în virgulă flotantă este reprezentat printr-un exponent **E** şi o mantisă **M**, mantisa întotdeauna subunitară şi normalizată (bitul cel mai semnificativ, cu excepţia bitului de semn, este diferit de zero). Valoarea numărului reprezentat în virgulă flotantă este $2^{\mathbf{E}-1}\cdot\mathbf{M}$.

Astfel, standardul IEEE 754 pentru numere în virgulă flotantă, prevede următoarele structuri pentru reprezentare pe 32 de biţi, respectiv 64 de biţi:

0	1	2	3	
MMMM MMMM	MMMM MMMM	EMMM MMMM	SEEE EEEE	Nr.∈ (±1.176E-38÷±3.40E+38)

S: bit de semn;

E: exponent, pe 8 biţi, în complement faţă de doi, cu deplasament 0x7Fh;

M: mantisă normalizată pe 23 de biţi; bitul cel mai semnificativ este întotdeauna 1 şi nu este memorat.

0	1	2	3	
MMMM MMMM	MMMM MMMM	MMMM MMMM	MMMM MMMM	
4	5	6	7	Nr.∈ (±1.7E-308÷±1.7E+308)
MMMM MMMM	MMMM MMMM	EEEE EMMM	SEEE EEEE	

S: bit de semn;

E: exponent, pe 12 biţi, în complement faţă de doi, cu deplasament 0x7FFh;

M: mantisă normalizată pe 51 de biţi; bitul cel mai semnificativ este întotdeauna 1 şi nu este memorat.

Adoptarea unui format nestandard, exponent 8 biţi şi mantisă 16 de biţi, permite utilizarea rutinelor de 16 biţi prezentate anterior (adunare, scădere, înmulţire, împărţire şi funcţiile trigonometrice care rezultă din sinus), implementarea algoritmului matematic trebuind să ţină cont de următoarele:

- adunarea şi scăderea se fac direct asupra mantisei, după ce una din acestea a fost denormalizată astfel încât cei doi exponenţi să fie egali;
- pentru înmulţire, după ce ambele mantise sunt normalizate, exponenţii se adună iar mantisele se înmulţesc;
- pentru împărţire, dacă ambele mantise sunt normalizate, exponenţii se scad iar mantisele se împart;
- funcţiile trigonometrice sunt deduse din sinus.

Rutinele de normalizare sau denormalizare nu sunt prezentate, ele fiind relativ simple: o deplasare la stânga a virgulei mantisei presupune o incrementare a exponentului, în timp ce o deplasare la dreapta a virgulei mantisei presupune o decrementare a exponentului.

2.6. Filtre numerice

În programele pentru achiziţie de date este uneori utilă implementarea unor filtre numerice, de exemplu rejecţia frecvenţei de 50 Hz.

În continuare se prezintă un program scris în C şi cu o funcţie scrisă în asamblare.

Conversia analog numerică se face folosind canalul 6 al microcontrolerului, pornirea conversiei făcându-se folosind un tact extern cu frecvenţa de 150.588Hz generat cu PWM1. La PWM0 este legat un LED verde prin care se semnalizează funcţionarea aparatului iar pe portul P4.0, P4.1 şi P4.2 sunt legate 3 LED-uri care semnalizează diverse nivele ale semnalului achiziţionat.

Filtrul numeric FIR proiectat este de tip FOB eliptic de ordin 4 cu banda de tăiere între 47Hz şi 53Hz cu pierderea maximă de 0.01dB în banda de trecere şi cu atenuarea minimă de 40dB în banda de tăiere. Coeficienţii acestuia au fost determinaţi în Matlab®, rezultând un filtru de ordin 4 cu caracteristica prezentată cu linie punctată în figura 3.7.

```
A = [1.0000000000000000e+000, 1.9339272398278690e+000,
     2.8285942321601420e+000, 1.8331973819885190e+000,
     8.9861884056193790e-001]

B = [9.4679436232257540e-001, 1.8813950274867980e+000,
     2.8281850745989090e+000, 1.8813950274867970e+000,
     9.4679436232257500e-001]
```

Deoarece operaţiile matematice folosind numere în virgula flotantă sunt foarte mari consumatoare de timp, vom implementa filtrul numeric folosind operaţii matematice pe numere întregi. Pentru menţinerea preciziei şi caracteristicii filtrului numeric calculat vom înmulţii coeficienţii cu o constantă şi după aceea îi vom rotunji la cel mai apropiat număr întreg pe 32 de biţi (`long int`). Constanta trebuie aleasă astfel încât să nu apară depăşire în calculele noastre considerând valoarea maximă a rezultatului convertorului analog numeric şi cea mai mare valoarea a coeficientului.

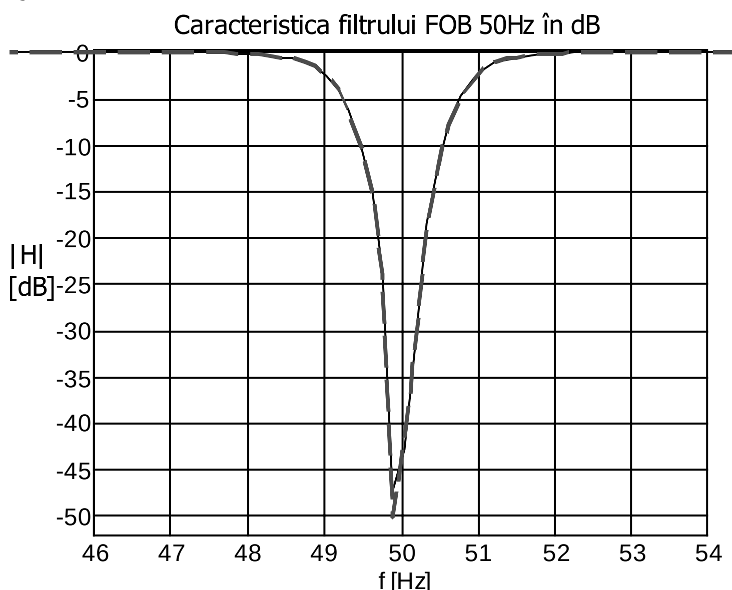


Figura 2.7 Caracteristica filtrului FIR eliptic FOB

Valoarea cea mai mare a coeficienţilor filtrului nostru este de aproximativ 3, valoarea maximă obţinută la ieşirea convertorului analog numeric este de 0x3FF=1.023 şi valoare ce mai mare care poate fi reprezentată pe 32 de biţi este de 0x7FFFFFFF= 2.147.483.647. Valorile intermediare obţinute în calculul filtrului sunt formate din adunarea a 4 înmulţiri. Astfel se poate determina

valoarea maximă a constantei: $c = \frac{0x7FFFFFFF}{4 \cdot 3 \cdot 0x3FF} = 0x2AB55$. Tot din considerente de viteză această constantă trebuie sa fie o putere a lui doi pentru că operaţiile de înmulţire şi împărţire pot fi implementate prin rotire la stânga şi respective la dreapta şi de asemenea nu trebuie să fie prea mare pentru limitarea numărului de rotiri. Noi am ales constanta la valoarea 0x4000. Filtrul obişnuit folosind coeficienţii multiplicaţi cu 0x4000 şi rotunjiţi la cel mai apropiat număr întreg are caracteristica prezentată în figura 3.7 cu linie continuă.

```
A = [16384, 31685, 46344, 30035, 14723];
B = [15512, 30825, 46337, 30825, 15512];
```

Funcţia de tratare a întreruperilor sosite de la convertorul analog numeric apelează o funcţie scrisă în asamblare care întoarce rezultatul conversiei pe 10 biţi într-un `unsigned int`.

Conform regulilor de transfer a valorilor returnate în R6 se găseşte partea MSB iar în R7 partea LSB.

Interfaţa serială RS232 a microcontrolerului este programată în 19200,8,N,1 şi este folosită pentru transferul eşantioanelor către PC.

La compilare se pot folosi două directive:

`IMPLEMENT_FILTER`: valorile eşantioanelor transmise sunt trecute prin filtrul numeric implementat. Dacă nu este specificată această directivă de compilare, eşantioanele trimise sunt cele obţinute de la convertorul analog numeric

`IMPLEMENT_DAC10BITS`: eşantioanele obţinute de la convertorul analog numeric sunt reprezentate pe 10 biţi şi dacă această directivă de compilare nu este specificată atunci rezultatul conversiei este pe numai 8 biţi (cei mai semnificativi).

```
/***********************************************************************\
**  Titlu:        FILTRU.C                                             **
**  Descriere:    Implementarea unui fltru numeric                     **
**                                                                     **
**  Versiune:     4.0                                                  **
**  Inceut la:    August 98                                            **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania   **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                   **
**                                                                     **
**  Acest modul contine urmatoarele:                                   **
**                Filtru numeric                                       **
**                Watchdog                                             **
**                Main                                                 **
**                                                                     **
**  Ultima modificare la data: 23 sep 1999                             **
**  Istoric:                                                           **
**                                                                     **
**  Observatii:                                                        **
**                Foloseste functia int getadc() scrisa in asamblare in **
**                fiserul ASMFUNC.ASM                                  **
\***********************************************************************/
#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
#include <Reg552.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include <absacc.h>
```

```
#include <typedef.h>

/* Canalul pe care se face achizitia de date */
#define CHANEL      6

/* Ordinul filtrului – trebuie sa fie putere a lui 2 */
#define FILTER_ORDER 4

;*****************************************************************
// OPTIUNI DE COMPILARE:
//     IMPLEMENT_FILTER    – achizitia de date este filtrata
//     IMPLEMENT_DAC10BITS – achizitia de date se face pe 10 biti
//                           altfel pe 8 biti
;*****************************************************************
//#define IMPLEMENT_FILTER 1
//#define IMPLEMENT_DAC10BITS     1

// Avem 3 LED-uri care se aprind in functie de nivelul semnalului
#ifdef IMPLEMENT_DAC10BITS
       #define       HIGH_DAC      999
       #define       LOW_DAC       870
#else
       #define       HIGH_DAC      0xf8
       #define       LOW_DAC       0xc8
#endif // IMPLEMENT_DAC10BITS

// rezultatul conversiei analog-numerice (este setat in intrerupere
// si citit in functia main)
data unsigned int DAC;

// indicator terminare converisie analog-numerica (setat in interupere
// si resetat in functia main)
bit fadc;

// Un sir de caractere de identificare
code char xx[] = "\nPRACON(R) V4.02 (c)StSoft 1999 - PRATCO s.r.l. Bucharest Romania.
CP 61-137 RO7550";

#ifdef IMPLEMENT_FILTER
// vector pt. implementarea listelor circulare necesare filtrului numeric
data long xfilter[FILTER_ORDER];
data long yfilter[FILTER_ORDER];
data char xi=0,yi=0;            // indexul in listele circulare

// Coeficientii filtrului numeric
// Pentru viteza nu se va lucra numai cu operatii pe numere intregi.
// Coeficientii au fost determinati in Matlab pentru un filtru opreste
// banda de ordin 4 cu frecventele de taiere [47Hz, 53Hz]. Coeficientii
// determinati au fost inmultiti cu 16384 si convertiti la intreg. S-a ales
// valoarea de 2^14=16384 (0x4000) pt. ca operatia de inmultire si impartire
// poate fi realizata prin rotire la stanga si respective la dreapta de 14
// ori.
data const long a[FILTER_ORDER + 1] = {16384, 31685, 46344, 30035, 14723};
data const long b[FILTER_ORDER + 1] = {15512, 30825, 46337, 30825, 15512};
#endif // IMPLEMENT_FILTER

// definitiile functiilor externe – din fiserul ASMFUNC.ASM
extern int getadc(void);

// definitiile functiilor C din acest fisier

// ==================================================================
// WATCHDOG
// ==================================================================
void watchdog(void)
{
   PCON |= 0x10;     // setam intai (PCON.4)
   T3 = 0x01;        // setam valoarea timerului watchdog (2*100ms)
}

#ifdef IMPLEMENT_FILTER
// ==================================================================
// FILTER = Filtru digital.
//     Y = FILTER(B, A, X) Filtreaza esantioanele prezentate
//     secvential in X folosind coeficientii filtrului prezentati in
//     vectorii A si B.
//     Metoda de calcul este:
//     y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
//                      - a(2)*y(n-1) - ... - a(na+1)*y(n-na)
// !!!! Din considerente de viteza ordinul filtrului trebuie sa fie
// !!!! o putere a lui 2 si operatiile se fac pe numere intregi.
// !!!! Pentru precizia calculelor coeficientii filtrului sunt inmultiti
// !!!! cu o constanta 2^14, iar rezulatutul este impartit cu aceeasi
```

```
// !!!! constanta. Daca constanta este o putere a lui 2 atunci operatiile
// !!!! de inmultire si impartire pot fi implementate folosind rotri la
// !!!! stanga si respective la dreapta.
// ===============================================================
long filter(long x)
{
   register char i=0;
   register long bx=0, ay=0;

   for(i = FILTER_ORDER; i > 0; i--)
   {
       bx += b[i] * xfilter[xi++ & (FILTER_ORDER - 1)];
       ay += a[i] * yfilter[yi++ & (FILTER_ORDER - 1)];
   }

   xfilter[xi++ & (FILTER_ORDER - 1)] = x;
   bx = (bx + b[0]*x - ay);
   bx /=  16384;
   //bx = bx >> 14; // impartim tot la 16384=0x4000=2^14

   yfilter[yi++ & (FILTER_ORDER - 1)] = bx;

   return(bx);
}
#endif // IMPLEMENT_FILTER

// ===============================================================
// MAIN routine
// ===============================================================
void main(void)
{
       long f=0;
       unsigned int old_val = 0;
       // long dummy=0;     // numarul de bucle facute in asteptare

       watchdog();

#ifdef IMPLEMENT_FILTER
       // Stergem listele circulare
       memset(xfilter, 0x00, sizeof(xfilter));
       memset(yfilter, 0x00, sizeof(yfilter));
#endif

// Aprindem tote LED-urile ON
       P4 = 0xff;           // [0xfe = yellow, 0xfd = red, 0xfb = green]

       // Setam interfata seriala la 19200,8,N,1
       EA = 0;              // invalidam toate intreruperile
       S0CON = 0x52;        // Setam interfata seriala
       TMOD = 0x20;
       PCON |= 0x80;
       TH1 = 0xfd;          // 19200, 8, N, 1
       TR1 = 1;
       ADCON = 0x00; // resetam ADCI, satrtare soft ADEX=0 ADCI=0 ADCS=0
       ADCON |= CHANEL;     // canalul care este folosit pt achizitie
       ADCON |= 0x08;       // setam ADCS (start ADC)

       fadc = 0;            // stergem indicatorul terminare achizitie
       EAD = 1;             // validam intreruperile de la ADC

       // Pentru generarea frecventei de achizitie folosim PWM1 cu factor
       // de umplere 50% si frecventa de 150.588Hz. Semnalul de la iesirea
// PWM1 este folosit pentru pornirea conversiei analog numerice
       // PWMP = 71; // folosim PWM1 ca sa generam 301.176Hz
       // PWMP = 216;       // folosim PWM1 ca sa generam 99.929Hz
       PWMP = 143;          // folosim PWM1 ca sa generam 150.588Hz
       PWM1 = 0x80;  // factor de umplere de 50% -> tact extern ADC

       // Iesirea PWM0 este folosita pentru aprinderea LED-ului POWER
       PWM0 = 0x00;         // POWER LED ON

       EA = 1;              // validam intruperile

       // puts(xx);  // trimitem mesajul de identificare

       ADCON |= 0x20;       // ADC START extern
       // ADCON |= 0x08     // ADC START soft

       // B U C L A   P R I N C I P A L A
       while(1)             //
       if(fadc)             // daca avem date de la convertorul ADC
       {
         fadc = 0;          // stergem indicatorul sfarsit conversie
```

```
#ifdef IMPLEMENT_FILTER
         f = filter(DAC);          // filtram esantioanele
#endif // IMPLEMENT_FILTER
         DAC &= 0x3fe;
          if(DAC > HIGH_DAC)
            P4 = 0xfb;             // GREEN LED
         else
              if(DAC < LOW_DAC)
               P4 = 0xfd;          // RED LED
              else
                P4 = 0xfe;         // YELLOW LED
#ifdef IMPLEMENT_FILTER
         printf("%ld\t", f);       // transmitem esantioanele filtrate
#else
         printf("%u\t", DAC);      // transmitem esantioanele
#endif // IMPLEMENT_FILTER
//       printf("%ld\t", dummy);   // cate bucle facem in asteptare
//       dummy = 0;                // anulam nr. De bucle
         watchdog();               // setam watchdog-ul
       }
//    else
//       dummy++;                  // incrementam nr de bucle
}

// ================================================================
//     Tratarea intreruperii de la ADC
// folosim bancul de registrii 2
// ================================================================
void INTADC(void) interrupt 10 using 2
{
#ifdef IMPLEMENT_DAC10BITS
       DAC = getadc();      // max = 0x3ff; citim toti cei 10 biti
#else
       DAC = ADCH;          // max = 0xff; citim numai cei 8 biti superiori
#endif
       ADCON &= 0xef;       // 1110 1111 => ADCI = 0
       ADCON |= CHANEL;     // set the aqusition chanel
       fadc = 1;
}

/*************************************************************************\
**  Titlu:        ASMFUNC.ASM                                            **
**  Descriere:    Citirea convertorului analog numeric                   **
**                                                                       **
**  Versiune:     4.0                                                    **
**  Inceut la:    August 98                                              **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania     **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                     **
**                                                                       **
**  Acest modul contine urmatoarele:                                     **
**                int getadc(void)                                       **
**                                                                       **
**  Ultima modificare la data: 23 sep 1999                               **
**  Istoric:                                                             **
**                                                                       **
**  Observatii:                                                          **
**                                                                       **
\*************************************************************************/
$XREF
$DEBUG
$NOMOD51
$INCLUDE (\INCLUDE\REG552.INC)

;_PROG     SEGMENT CODE
_GETADC    SEGMENT CODE
;BYTEVAR   SEGMENT DATA BITADDRESSABLE
;VAR       SEGMENT DATA
;BITVAR    SEGMENT BIT
;STACK     SEGMENT IDATA
;CONST     SEGMENT CODE

;================================================================
; int getadc(void)
; un intreg este returnat folosin registrii R6(MSB) si R7(LSB)
;================================================================
PUBLIC     GETADC
           RSEG       _GETADC

GETADC:
           mov        a,ADCH          ;citim rezultatul conversisi
           swap       A               ;schimbam high cu low din A
           rr         A
           rr         A
```

```
          anl       A,#03h          ;mascam ADC9 si ADC8
          mov       R6,A            ;partea MSB a lui int [max = 0x03]
          mov       A,ADCH
          anl       A,#3fh
          rl        A
          rl        A
          mov       R7,A            ;partea LSB a lui int
          mov       A,ADCON         ;low part from ADC
          rlc       A
          rlc       A
          rlc       A
          anl       A,#03h
          orl       A,R7            ; partea LSB a lui int
          mov       R7,A
          ret
;=================================================================
END
```

În continuare este prezentat programul MATLAB cu care s-a proiectat filtrul FIR FOB prezentat anterior.

```
fs=150.588;                              % frecventa de esantionare Hz
N=512;                                   %nr. de puncte pt desenarea graficelor
%proiectam un FOB cu fl=47Hz si fh=53Hz
fl=47;
fh=53;
Wl=fl*2/fs;
Wh=fh*2/fs;
Rp=0.01;                                 % max. lost in passband [dB]
Rs=40;                                   % attenuation in stopband [dB]
Ne=2;                                    % ordinul filtrului eliptic
% proiectam un FOB eliptic
[Be, Ae] = ellip(Ne, Rp, Rs, [Wl Wh], 'stop');
%Be = fir1(33, [Wl Wh], 'stop'); Ae = [1 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
Be = round(Be*16384)
Ae = round(Ae*16384)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[He, We] = freqz(Be, Ae, N);
%plot(We*fs/(2*pi), abs(He), 'r-'); grid on;
plot(We*fs/(2*pi), abs(He), 'r-'); grid on;
title('Caracteristica filtrului FOB 50Hz');
xlabel('Hz');
ylabel('A');

figure;
fx = We*fs/(2*pi);
ix = find(fx>50);

plot(fx(ix-40:ix+40), 20*log10(abs(He(ix-40:ix+40))), 'r-'); grid on; hold on;
%plot(fx, 20*log10(abs(He)), 'r-'); grid on; hold off;
title('Caracteristica filtrului FOB 50Hz in dB');
xlabel('f [Hz]');
ylabel('|H| [dB]');

t=(1:150)/fs;
s1=sin(2*pi*t*15);
s2=sin(2*pi*t*20);
s3=sin(2*pi*t*45);
s4=sin(2*pi*t*50);
s5=sin(2*pi*t*55);
s=round((s1+s2+s3+s4+s5)*1023);
%figure;
%plot(t,s);
sf=round(filter(Be, Ae, s));
%sf=filter(Be, Ae, s);

figure;
plot(t,s,t,sf);

S=fft(s,512);
SF=fft(sf,512);
W=(0:255)/256*fs/2;
figure;
plot(W, abs(S(1:256)), 'r-', W, abs(SF(1:256)), 'g-'); grid on;
title('Filtrarea unui semnal');
xlabel('Hz');
ylabel('|FFT|');
```

2.7. Ceas de timp real

Există multe metode şi circuite care pot fi folosite în implementarea unui ceas de timp real.

Ce mai simplă metodă este folosirea unui timer intern microcontrolerului programarea acestui ca să genereze întreruperi la un interval de timp prestabilit. Această metodă are avantajul că nu necesită componente auxiliare şi este foarte uşor de implementat dar principalele dezavantaje sunt că rutina necesară implementării calendarului şi ceasului este destul de lungă şi deci microcontrolerul trebuie să consume mult timp cu acest proces. Un alt dezavantaj este că atunci când procesorul se află în modul power management timerul nu funcţionează, şi implementarea unui sistem de back-up a microcontrolerului este neeconomică datorită consumului ridicat al acestuia împreună cu circuitele aferente (RAM, EPROM, latch-uri etc.) în comparaţie cu consumul unui circuit integrat specializat.

Pentru implementarea unui ceas de timp real se poate folosi timerul T1 care este utilizat pentru generarea tactului pentru interfaţa serială, sau timerul T0 care poate fi programat pentru generarea de întreruperi între 750ns şi 192µs (oscilator 16MHz şi 8xC552).

```
/*********************************************************************\
**  Titlu:         TIMER0.H                                          **
**  Descriere:     Folosirea Timerului T0 ca generator de tic-uri    **
**                 Implementarea unui ceas de timp real              **
**                 Declaratii de constante si functii                **
**                                                                   **
**  Versiune:      2.0                                               **
**  Inceut la:     August 95                                         **
**  Autor:         Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                 **
**                                                                   **
**  Acest modul contine urmatoarele:                                 **
**                 Definerea tiplului TTIME                          **
**                 Declararea variabilei totic ca contor de ticuri   **
**                 Declaratiile functiilor:                          **
**                    bit T0_int(long tic) = inițializare T0         **
**                    TTIME T0_ReadTime() = macro citire contor tic-uri **
**                    TTIME T0_DiffTime(TTIME otic) = calculeaza diferenta **
**                            intre doua valori TTIME                **
**                                                                   **
**  Ultima modificare la data: 23 nov 1998                           **
**  Istoric:                                                         **
**                                                                   **
**  Observatii:                                                      **
**                                                                   **
\*********************************************************************/
#ifndef _TIMER0_H
#define _TIMER0_H   1

#include <typedef.h>

/* Tipul variabilei care va mentine cotorul de tic-uri [byte pt. viteza]*/
typedef byte TTIME;
/* Contorul de ticuri se va tine in 'data' pentru viteza */
extern data TTIME t0tic;

/* initializare timer 0 cu tic=[750ns...192us], ret=1->eroare */
bit T0_init(long tic);

/* Citeste contorul de ticuri */
#define      T0_ReadTime()      (tic)

/* Calculeaza diferenta de timp in tic-uri */
TTIME T0_DiffTime(TTIME otic);
#endif

/*********************************************************************\
**  Titlu:         TIMER0.C                                          **
```

```
**  Descriere:    Folosirea Timerului T0 ca generator de tic-uri          **
**                Implementarea unui ceas de timp real                    **
**                Declaratii de constante si functii                      **
\*************************************************************************/
#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
/* #pragma REGISTER BANK(0) */
#include <reg552.h>
#include <stdio.h>
#include <absacc.h>
#include "timer0.h"

/* comenteaza linia urmatoare pt introducere program de test*/
#define NO_MAIN     1

/* la cate ticuri se va executa functia utilizator [valoare = 2**n]*/
#define DO_T0_USRFNCT      0x7f

/* definire valorile minime si maxime ale intervalului de timp
   intre 2 tic-uri in ns */
#define MIN_TIC     750l
#define MAX_TIC     192000l

data TTIME t0tic;          /* contor de tic-uri */
extern void on_T0tic(void);/* functie utilizator apelata la fiecare tic */
bit fonT0tic;              /* flag tratare functie utilizator externa */

\************************************************************************
  int_T0(void) = Functia care este apelata in intreruperi pentru T0
                 Foloseste bancul 3 de registrii
************************************************************************/
void int_T0(void) interrupt 1 using 3
{
  if(!((t0tic++) && DO_T0_USRFNCT)) /* din cand in cand apeleaza fnct usr */
    if(!fonT0tic)   /* ? apel inhibat ? */
    {
      fonT0tic = 1;  /* inhibam eventualele apeluri ale functiei on_T0tic  */
             /* daca timpul de executie al acesteia este mai mare  */
                /* decat intervalul dintre 2 intreruperi              */
      on_T0tic();   /* apelam functia utilizator                       */
      fonT0tic = 0;  /* validam apelurile ulterioare ale functiei on_T0tic */
    }

}

\************************************************************************
  Program de test
************************************************************************/
#ifndef NO_MAIN
main()
{
       TTIME Start,xtic;

       EA = 0;                     /* DISABLE ALL INTERRUPTS */
       /* PENTRU QUARTZ DE 16 MHz, 9600 BAUDS */
       SCON = 0x52;         /* SCON */    /* setup serial port control */
       TMOD = 0x20;         /* TMOD */    /* hardware (9600 BAUD @16MHZ)*/
       PCON = PCON | 0x80;  /* SMOD = 1 */
       TH1 =  0xf7;         /* TH1 */
       TR1 = 1;             /* Timer 1 Start */
       EA = 1;                     /* ENABLE ALL INTERRUPTS */

       /* Setare Timer 0 la 100000ns = 100us = 0.1s */
       if(T0_init(100000))
         printf("Eroare initializare T0 la 0.1ms\n");

       Start = T0_ReadTime();

       while(1)
       if((xtic=T0_DiffTime(Start)) > 10)
       {
         Start = T0_ReadTime();
         printf("TIC %u\n", xtic);
       }
}
#endif

\************************************************************************
 T0_DiffTime(TTIME otic) = calculeaza diferenta intre 2 valori de tic
************************************************************************/
TTIME T0_DiffTime(TTIME otic)
{
/*    return 0; */
  return((t0tic >= otic) ? (t0tic-otic) : (0xff-otic+t0tic));
```

```
/*  return(T0_ReadTime() - otic); */
}

\*******************************************************************
  bit T0_init(long tic) = intilaizeaza Timerul T0 cu valoarea din argument
                        in ns
*******************************************************************/
bit T0_init(long tic)
{
      data int x;

      /* tic in ns = [750ns,...,192000ns=192us]
      /* Fosc = 16Mhz/12 = 1.333.333,333                          */
      /* tic = (256-TH0)/Fosc                               */
      /* tic min = (256-255)/Fosc = 750ns          (MIN_TIC)            */
      /* tic max = (256-000)/Fosc = 192000ns=192us (MAX_TIC) */

      /* verificam domeniul de valabilitate pt. tic in [ns] */
      if((tic < MIN_TIC) || (x > MAX_TIC))
        return 1;          /* intoarcem ca aparut o eroare */

      x = 256-tic/MIN_TIC;

      EA = 0;                    /* DISABLE ALL INTERRUPTS */
       fonT0tic = 0;
      /* Setare Timer 0 */
      TMOD = TMOD | 0x02;  /* Timer 0 auto reload */
      TH0 = (char)x;
      TR0 = 1;             /* start timer 0 */
      ET0 = 1;             /* enable timer 0 interrupt */
      EA = 1;                    /* ENABLE ALL INTERRUPTS */
      return 0;
}
```

Modul de programare şi de folosire al timerului `T0` este prezentat în programul de mai sus. Pornirea timerului `T0` se face prin apelul funcţiei `T0_init` care are ca parametru intervalul de timp dorit între 2 întreruperi. Această valoare este mai întâi verificată dacă se încadrează în limitele admise pentru frecvenţa oscilatorului. Dacă oscilatorul are o frecvenţă diferită de 16MHz, trebuiesc recalculate valorile minime şi maxime (`MIN_TIC` şi `MAX_TIC`) şi înlocuirea acestora în fişierul `TIMER0.H`.

Funcţia care se apelează în întreruperi incrementează valoarea globală a contorului de ticuri `T0tic` care este implementat ca `unsigned char` în zona de memorie data din considerente de viteză. Valoarea contorului este verificată pentru a se vedea dacă se poate apela funcţia utilizator. Această funcţie trebuie să se numească `void on_T0tic(void)` şi dacă aceasta lipseşte trebuie modificată funcţia `int_T0` apelată în întreruperi. Pentru inhibarea reapelului funcţiei utilizator înainte de terminarea acesteia (funcţia poate să nu fie reentrantă) este implementată o interblocare folosind indicatorul `fonT0tic`.

A doua metodă de implementare a unui ceas de timp real, prezentată în continuare, foloseşte un circuit specializat RTC72421. Acesta este văzut ca un periferic în spaţiul de adresare al microcontrolerului. Comunicaţia între RTC şi microcontroler se face pe 4 biţi (`D0÷D3`). Acest circuit conţine un oscilator propriu şi poate să furnizeze ora şi data. De asemenea acest circuit poate fi programat să genereze şi întreruperi la un anumit interval de timp cât şi la o anumită oră (alarmă).

Registrele interne ale acestui circuit sunt definite în fişierul `SYSTEM.H` din anexe şi au descrierea din tabelul 3.12.

Tabelul 2.14

Adresa					Data				Nume registru	Gamă valoare
Hexa	A3	A2	A1	A0	D3	D2	D1	D0		
0	0	0	0	0	S8	S4	S2	S1	SEC01	0÷9
1	0	0	0	1	-	S40	S20	S10	SEC10	0÷5
2	0	0	1	0	M8	M4	M2	M1	MIN01	0÷9
3	0	0	1	1	-	M40	M20	M10	MIM10	0÷5
4	0	1	0	0	H8	H4	H2	H1	HOUR01	0÷9
5	0	1	0	1	-	PM/$\overline{AM}$	H20	H10	HOUR10	0÷2
6	0	1	1	0	D8	D4	D2	D1	DAY01	0÷9
7	0	1	1	1	-	-	D20	D10	DAY10	0÷3
8	1	0	0	0	M8	M4	M2	M1	MON01	0÷9
9	1	0	0	1	-	-	-	M10	MON10	0÷1
A	1	0	1	0	Y8	Y4	Y2	Y1	YEAR01	0÷9
B	1	0	1	1	Y80	Y40	Y20	Y10	YEAR10	0÷9
C	1	1	0	0	-	W4	W2	W1	WEEK	0÷6
D	1	1	0	1	30 adj	IRQ flag	BUSY	HOLD	REG D	-
E	1	1	1	0	t1	t0	Intr/$\overline{Stnd}$	MASK	REG E	-
F	1	1	1	1	TEST	24/$\overline{12}$	STOP	RESET	REG F	-

Observaţii:

1. Biţii marcaţi cu '-' nu există şi nu contează la scriere;
2. Bitul PM/$\overline{AM}$ trebuie mascat în timpul citirii zecilor de ore;
3. Bitul BUSY poate fi numai citit;
4. Bitul IRQ flag poate fi setat numai la valoarea 0;
5. Calendarul este setat pentru era creştină;
6. Zilele săptămânii sunt memorate astfel: 0=duminică … 6=sâmbătă;
7. La scrierea unei valori într-un registru care este în afara limitelor acestuia va rezulta o eroare la citire.

Pinii D0÷D3 (Data Bus) sunt folosiţi pentru citirea şi scrierea datelor în circuit şi sunt conectaţi la magistrala de date. Când $\overline{CS0}$ = '0', CS1 = '1' şi sunt activaţi pinii RD sau WR putem citi respectiv scrie data în circuit, în rest ei sunt în starea de impedanţă ridicată (3-state).

Pinii A0÷A3 sunt pinii de adresă şi sunt folosiţi împreună cu pinul ALE.

Pinul ALE validează decodorul intern de adrese. Când ALE='1' şi $\overline{CS0}$='0', data care se găseşte pe pinii A0÷A3 sunt scrişi în decodorul de adrese. Microprocesoarele cu ieşirea ALE poate fi conectată direct la acest pin, pentru celelalte procesoare ALE trebuie legat la V_{cc}.

Pinul $\overline{WR}$ este folosit pentru a valida porţile care permit scrierea datelor în registrul desemnat de A0÷A3.

Pinul $\overline{RD}$ este folosit pentru a valida porţile care permit citirea datelor în registrul desemnat de A0÷A3.

Pinii $\overline{CS0}$ **şi CS1** sunt folosiţi pentru selecţia circuitului şi validează interpretarea semnalele ALE, $\overline{RD}$ şi $\overline{WR}$ ($\overline{CS0}$ = '0', CS1 = '1'). CS1 funcţionează separat de ALE şi trebuie folosit pentru detecţia alimentării circuitului.

Pinul STD.P (*Standard Pulse*) este folosit pentru generarea de întreruperi şi este controlat de registrul E (REG E). Ieşirea este open-drain şi

nu depinde de $\overline{CS0}$ şi CS1. Valoarea rezistenţei sarcinii legate la V_{cc} trebuie să fie de minimum 2K2.

Registrul de control D

HOLD (D0). Setând acest bit la valoarea 1 LOGIC se validează trecerea în 0 LOGIC a bitului BUSY. Când bitul BUSY este şters, registrele RTC pot fi citite sau scrise. După terminarea operaţiei de scriere sau citire dintr-un registru, bitul HOLD trebuie resetat. Dacă acest lucru nu este făcut va apare o eroare. Dacă în timpul în care HOLD este 1 LOGIC RTC-ul trebuie să incrementeze registrul de secunde S1, acesta va fi incrementat la o secundă după ce bitul HOLD redevine 0 LOGIC (HOLD trebuie să fie setat mai puţin de o secundă). Dacă CS1 = 0 LOGIC, atunci HOLD = 0 LOGIC.

BUSY (D1) este folosit pentru a determina dacă registrele de la S1 la W pot fi accesaţi pentru scriere sau citire, acest lucru fiind posibil numai atunci când bitul BUSY ='0'. Acest bit poate fi numai citit (*read only*). Pentru a accesa registrele RTC de la S1 la W este indicat să se respecte algoritmul prezentat în figura 3.8.

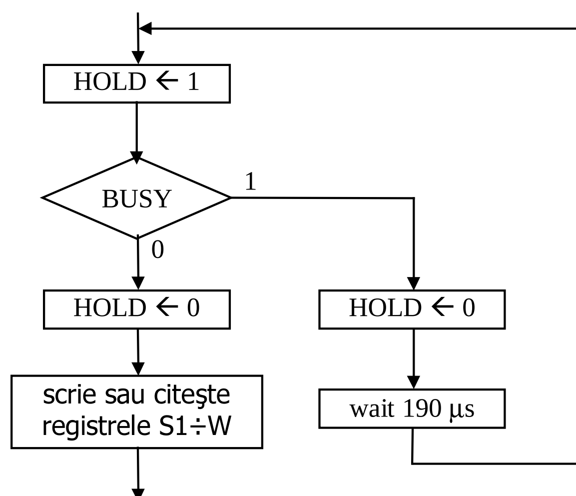


Figura 2.8. Diagrama de utilizare a RTC 72421

IRQ flag (D2). Acest bit (*Interrupt Request Flag*) este controlat direct de pinul de ieşire STD.P. Când STD.P este şters, IRQ=1 LOGIC şi atunci când STD.P este setat, IRQ=0 LOGIC. Când IRQ ='1' atunci este cerută o întrerupere microcontrolerului. Bitul MASK împreună cu t0 şi t1 şi INTR/STND controlează intervalul de timp pentru generarea pulsurilor la pinul STD.P. Bitul MASK are rolul de ON/OFF, INTR/STND controlează forma de undă şi t0 împreună cu t1 controlează perioada.

Când RTC-ul este în modul INTERRUPT, ieşirea STD.P va rămâne în starea '0' până când se va scrie '0' în bitul IRQ. Când RTC-ul este în modul STANDARD pinul STD.P va rămâne în starea '0' până când se va reseta manual bitul IRQ dar nu mai mult de 7.8125 ms atunci când bitul IRQ se va reseta automat.

Atunci când IRQ este şters şi vine o nouă întrerupere, noua întrerupere va fi ignorată. La setarea bitului HOLD sau 30 SEC ADJUST este necesară mascarea cu '1' a bitului IRQ. Setarea bitului IRQ la valoarea '1' nu are nici un efect pentru pinul de ieşire STD.P.

Când se modifică starea biţilor t0 şi t1, bitul IRQ trebuie resetat.

30 SEC ADJ (D3). La setarea acestui bit se ajustează registrele secundelor la '00', conform tabelului 3.13. Acest bit se va reseta automat după 76.3μs. Cât timp acest bit este setat, registrele RTC nu pot fi accesate.

Tabelul 2.15

Valoarea sec. înainte de ajustare	Incrementare minute	Valoarea sec. după ajustare
30÷ 59	DA	00
01 ÷ 29	NU	00

Registrul de control E

MASK (D0). Acest bit lucrează ca întrerupător ON/OFF pentru ieşirea STD.P. Când bitul MASK ='1' atunci ieşirea STD.P este activă iar atunci când bitul MASK = '0' atunci ieşirea STD.P este în starea '1'.

INTR/$\overline{\text{STND}}$ (D1) este folosit pentru controlul formei de undă obţinute la ieşirea STD.P (open-drain). În modul INTERRUPT (D1 = '1') ieşirea STD.P va rămâne în starea '0' până când se va reseta manual bitul IRQ flag. În modul STANDARD ieşirea va rămâne de asemenea în starea '0' până la resetarea manuală a bitului IRQ flag dar nu mai mult de 7.8125ms când indicatorul IRQ se va reseta automat. Bitul MASK trebuie să fie '0' în timpul acestor moduri de lucru. Perioada semnalului de ieşire la pinul STD.P este determinată de biţii t0 şi t1.

t0 (D2) şi **t1** (D3) determină perioada semnalului de la ieşirea STD.P în cele două moduri de funcţionare INTERRUPT sau STANDARD, conform tabelului 3.14.

Tabelul 2.16

t1	t0	Perioada	Lăţimea impulsului	
			INTR	STND
0	0	1/64 sec	*	7.8125ms
0	1	1 sec	*	7.8125ms
1	0	1 minut	*	7.8125ms
1	1	1 oră	*	7.8125ms

ATENŢIE Lăţimea impulsului depinde de momentul resetării bitului IRQ flag. În momentul setării biţilor t0 şi t1 începe o perioadă. Biţii t0 şi t1 nu sunt asociaţi nici unui contor.

Registrul de control F

RESET (D0). Acest bit este utilizat pentru resetarea contorilor interni cu frecvenţa mai mică de 1Hz. La reset nu vor fi afectaţi registrele de ceas sau dată. Atâta timp cât RESET = '1' numărătoarele vor fi oprite. Pentru a reporni

numărătoarele, bitul RESET trebuie resetat la valoare '0'. Dacă CS1 = '0' atunci RESET = '0'.

STOP (D1). Când bitul STOP este '1' va opri reţeaua de divizare a frecvenţei de 8192Hz, iar când este '0' RTC-ul va funcţiona normal. Există o întârziere de 122µs după schimbarea stării acestui bit.

24/$\overline{12}$ (D2). Acest bit este utilizat pentru selectarea modului de indicare a orei în format 24 (când este setat la valoarea '1') sau 12 (când este setată valoarea '0'). În modul 24 ore, trebuie ignorată valoarea bitului PM/$\overline{AM}$.

TEST (D3). Acest bit este folosit pentru testare şi trebuie setat la valoarea '0' pentru ca RTC-ul să funcţioneze normal.

Observaţii

1. În timpul ajustării la 30 secunde (30 sec adj.) poate apare un impuls la pinul STD.P.

   Ieşirea STD.P îşi va menţine starea în care este când bitul STOP = '1' atunci când INTR/$\overline{STND}$ = '0' (modul STANDARD).
2. Nu se va produce nici o modificare a ieşirii STD.P ca urmare a scrierii în registrele de la S1 la H1.
3. Un semnal mai mare de 4/5V_{DD} trebuie aplicat pe pinul CS1 pentru circuitul să fie activat şi trebuie să fie mai mic de 1/5 V_{DD} pentru a nu apare un curent rezidual nedorit.
4. Pentru protecţia datelor la punerea sub tensiune, CS1 trebuie să fie activat după cel puţin 2µs, iar la anularea tensiunii, CS1 trebuie trecut în starea '0' cu cel puţin 2µs înainte

Iniţializarea şi setarea alarmei trebuie făcută conform organigramei din figura 3.9.

Funcţiile prezentate în continuare permit setarea, citirea şi scrierea din/în circuitul integrat RTC-72421. Explicaţiile din cod, împreună cu descrierea registrelor circuitului din paragraful anterior, sunt suficiente pentru a înţelege codul prezentat în continuare. Trebuie menţionat că în întreruperea RTC au fost implementate şi două timere care vor fi decrementate până la anularea valorii setate în prealabil în variabilele secF1 şi/sau secF2, moment în care se vor seta indicatorii fF1 şi respectiv fF2.

Programele sursă folosesc fişierele de definiţii SYSTEM.H şi TYPEDEF.H care sunt prezentate în anexă.

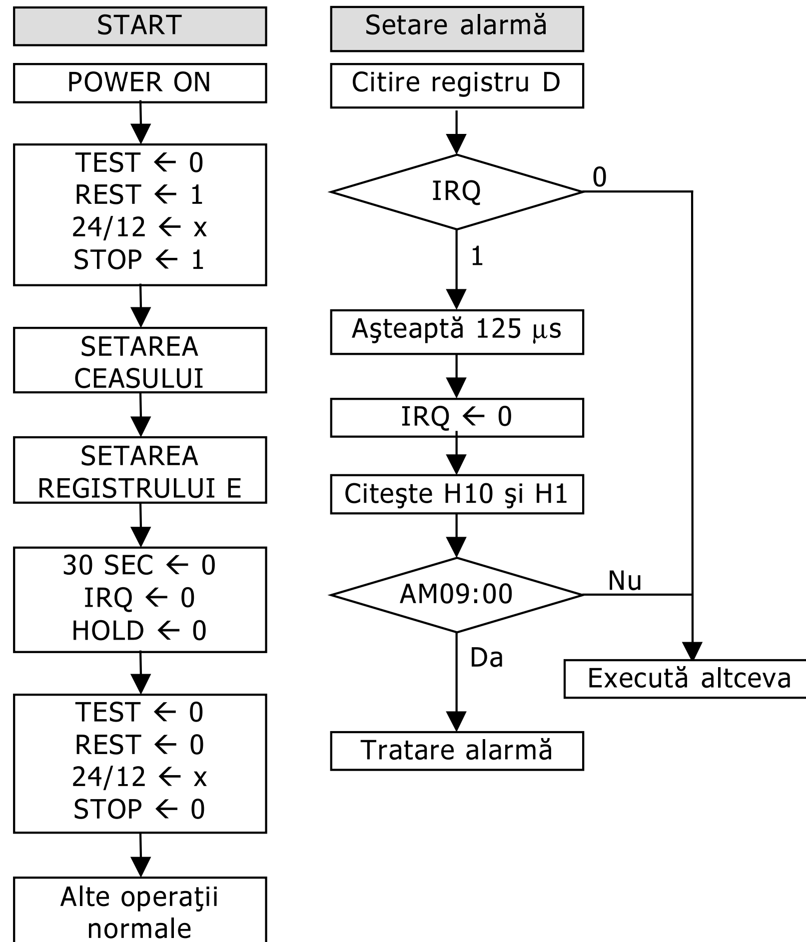


Figura 2.9. Programarea RTC72421

```
/***********************************************************************\
**  Titlu:         RTC.H                                                **
**  Descriere:     Descrierea functiilor pt. manipularea RTC-72421      **
**                                                                      **
**  Versiune:      2.0                                                  **
**  Inceut la:     August 95                                            **
**  Autor:         Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania   **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                    **
**                                                                      **
**  Acest modul contine urmatoarele:                                    **
**                 Functii pentru citirea si setarea RTC-ului           **
**                 Definirea unui sir de caractere ASCII cu data ora ... **
**                                                                      **
**  Ultima modificare la data: 23 nov 1998                              **
**  Istoric:                                                            **
**                                                                      **
**  Observatii:                                                         **
**                                                                      **
\***********************************************************************/
#ifndef      _RTC_H_
#define      _RTC_H_            1

extern xdata char    RTC_DATA[22];
#define              RTC_DATE     RTC_DATA + 2
#define              RTC_TIME     RTC_DATA + 11
#define              RTC_WEEK     RTC_DATA[20]
#define              RTC_WEEK_STR RTC_DATA

extern code char days[8][3];

extern bit    fRTC;         // Flag intrerupere RTC (1s)
extern bit    fTIC;         // Flag intrerupere RTC (1/64s = 15.625ms)
extern bit    fF1;          // Flag functie speciala (1 dupa secF1 * 1/64s)
extern bit    fF2;          // Flag functie speciala (1 dupa secF2 * 1/64s)
extern idata word tic;               // contor de ticuri (++ la fiecare 15.625ms)
extern idata word secF1;    // dupa cate 1/64s se seteaza fF1
extern idata word secF2;    // dupa cate 1/64s se seteaza fF2

#define GetTic()     tic
#define TicDiff(otic)      ((tic >= otic) ? (tic-otic) : (0xffff-otic+tic))
```

```
extern bit RTC_busy(void);              // citire bit BUSY
extern void RTC_init(void);             // initializare RTC
extern void RTC_clr(void);              // sterger date RTC
extern void RTC_read(void);             // citire date RTC

extern void setRTCday(byte val);        // setare zi
extern void setRTCmon(byte val);        // setare luna
extern void setRTCyear(byte val);       // setare an
extern void setRTCweek(byte val);       // setare zi din saptamana
extern void setRTChour(byte val);       // setare ora
extern void setRTCmin(byte val);        // setare minute
extern void setRTCsec(byte val);        // setare secunde

#endif

/***************************************************************************\
**  Titlu:        RTC.C                                                  **
**  Descriere:    Implementarea functiilor pt. manipularea RTC-72421     **
**                                                                       **
**  Acest modul contine urmatoarele:                                     **
**                Functii pentru citirea si setarea RTC-ului             **
**                Definirea unui sir de caractere ASCII cu data ora ...  **
\***************************************************************************/
#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
#include <reg552.h>
#include <string.h>
#include <typedef.h>
#include <absacc.h>
#include "i2c.h"
#include "qcount.h"
#include "..\main\system.h"

xdata char   RTC_DATA[22];        // “dd/mm/yy hh:mm:ss ww”
code char days[8][3] = {"Du", "Lu", "Ma", "Mi", "Jo", "Vi", "Sa", "??"};

bit    fRTC;         // Flag intrerupere RTC (1s)
bit    fTIC;         // Flag intrerupere RTC (1/64s = 15.625ms)
bit    fF1;          // Flag inerval de timp scurs 1
bit    fF2;          // Flag inerval de timp scurs 2
//bit  onTicRTC;     // suntem in apelul functiei ticRTC
idata word tic;      // incrementat de 1/64s (15.625ms)
idata word secF1=0;  // itervalul de timp in 1/64s pentru setarea lui fF1
idata word secF2=0;  // itervalul de timp in 1/64s pentru setarea lui fF2

bit RTC_busy(void);  // citire bit BUSY
void RTC_init(void); // initializare RTC
void RTC_clr(void);  // stergere date din RTC
void RTC_read(void); // citire RTC

extern idata byte RTC_ck;  // folosit pentru verificarea functinarii RTC-ului

\*************************************************************************
  Tratarea intreruperilor de la RTC
*************************************************************************/
void int_rtc(void) interrupt 2 using 1
{
  tic++;             // contor ticuri
  fTIC = 1;          // TIC la fiecare 1/64sec

  // contor de intarziere 1
  if(secF1)
    if(!(--secF1))   // decrementam timer 1 pana la ‘0’
      fF1 = 1;              // a ajuns la zero -> setam flagul fF1

  // contor de intarziere 2
  if(secF2)
    if(!(--secF2))   // decrementam timer 2 pana la ‘0’
      fF2 = 1;              // a ajuns la zero -> setam flagul fF2

  if(!(tic && 0x3f))        // echivalent cu if(!(tic % 64))
    fRTC = 1;        // flag 1 sec

  RTC_ck = 2;               // supraveghere functionare RTC cu timer 0
}

\*************************************************************************
  Citim datele din RTC si le formatam sub forma unui sir ASCIIZ cu urmatoarea
  structra: dd/mm/yy hh:mm:ss ww.
                     ^     ^
                     |     +-- se inscrie ‘\0’
                     +-------- din secunda in secunda se va inlocui cu ‘ ‘
*************************************************************************/
```

```
void RTC_read(void)
{
  while(RTC_busy());

  RTC_DATA[2]  = (XBYTE[regDAY10] & 0x03) + '0';      // d10
  RTC_DATA[3]  = (XBYTE[regDAY1]  & 0x0f) + '0';      // d1
  RTC_DATA[4]  = '/';
  RTC_DATA[5]  = (XBYTE[regMON10] & 0x01) + '0';      // m10
  RTC_DATA[6]  = (XBYTE[regMON1]  & 0x0f) + '0';      // m1
  RTC_DATA[7]  = '/';
  RTC_DATA[8]  = (XBYTE[regYEAR10] & 0x0f) + '0';     // y10
  RTC_DATA[9]  = (XBYTE[regYEAR1]  & 0x0f) + '0';     // y1
  RTC_DATA[10] = ' ';
  RTC_DATA[11] = (XBYTE[regHOUR10] & 0x03) + '0';     // h10
  RTC_DATA[12] = (XBYTE[regHOUR1]  & 0x0f) + '0';     // h1
//  RTC_DATA[13] = ':';
  RTC_DATA[13] = (XBYTE[regSEC1] & 0x01) ? ':' : ' ';
  RTC_DATA[14] = (XBYTE[regMIN10] & 0x07) + '0';      // m10
  RTC_DATA[15] = (XBYTE[regMIN1]  & 0x0f) + '0';      // m1
  RTC_DATA[16] = ':';
  RTC_DATA[17] = (XBYTE[regSEC10] & 0x07) + '0';      // s10
  RTC_DATA[18] = (XBYTE[regSEC1]  & 0x0f) + '0';      // s1
  RTC_DATA[19] = 0;
//  RTC_DATA[19] = ‘ ‘;
  RTC_DATA[20] = XBYTE[regWEEK] & 0x07;
  memcpy(RTC_DATA, days[RTC_DATA[20]], 2);     // ww

  XBYTE[regD_RTC] = 0x00;
}

\**************************************************************************
  Citirea bitului BUSY. Functa intoarce 1 cand RTC este ocupat si 0 in rest.
**************************************************************************/
bit RTC_busy(void)
{
  // HOLD = 1
  XBYTE[regD_RTC] = 0x01;  // 30adj=0,irq=1,busy=0,hold=1 (0101)

  if(XBYTE[regD_RTC] & 0x02)
  {
    // HOLD = 0
    XBYTE[regD_RTC] = 0x00;// 30adj=0,irq=1,busy=0,hold=0 (0100)
    return 1;              // trebuie sa asteptam
  }
  else
    return 0;              // RTC ramane in HOLD
}

\**************************************************************************
  Initializarea RTC-ului
**************************************************************************/
void RTC_init(void)
{
  IT1 = 1;                 // INT1 on falling edge
  EX1 = 1;                 // Enable extarnal interrupt 1 (RTC)

  XBYTE[regF_RTC] = 0x06;  // test=0,24h=1,stop=1,reset=0 (0110)

  XBYTE[regE_RTC] = 0x00;  // INT=1/64s(00), STND=1(0), MASK=0 (0000)
//  XBYTE[regE_RTC] = 0x04;// INT=1sec(01), STND=1(0), MASK=0 (0100)
//  XBYTE[regE_RTC] = 0x08;// INT=1min(10), STND=1(0), MASK=0 (1000)
//  XBYTE[regE_RTC] = 0x0c;// INT=1hour(11), STND=1(0), MASK=0 (1100)
  XBYTE[regD_RTC] = 0x00;  // 30adj=0, irq=0, busy=0, hold=0 (0000)
  XBYTE[regF_RTC] = 0x04;  // test=0, 24h=1, stop=0, reset=0 (0100)

  strcpy(RTC_DATA, "wwdd/mm/yy hh:mm:ss w");
//                  012345678901234567890

//  secF1 = 60;            // implicit fF1 setat din minut in minut
//  secF2 = 180;           // implicit fF2 setat din 3 minute in 3 minute
}

\**************************************************************************
  Stergerea datelor din RTC
**************************************************************************/
void RTC_clr(void)
{
  char adr;

  IT1 = 1;                 // INT1 on falling edge
  EX1 = 1;                 // Enable extarnal interrupt 1 (RTC)

  XBYTE[regF_RTC] = 0x07;  // test=0,24h=1,stop=1,reset=1 (0111)
```

```
  for(adr=0; adr<0x0d; adr++)
     XBYTE[RTC_ADR + adr] = 0x00;

//  XBYTE[regE_RTC] = 0x00;// INT=1/64s(00), STND=1(0), MASK=0 (0000)
  XBYTE[regE_RTC] = 0x04;  // INT=1sec(01), STND=1(0), MASK=0 (0100)
//  XBYTE[regE_RTC] = 0x08;// INT=1min(10), STND=1(0), MASK=0 (1000)
//  XBYTE[regE_RTC] = 0x0c;// INT=1hour(11), STND=1(0), MASK=0 (1100)
  XBYTE[regD_RTC] = 0x00;  // 30adj=0, irq=0, busy=0, hold=0 (0000)
  XBYTE[regF_RTC] = 0x04;  // test=0, 24h=1, stop=0, reset=0 (0100)
}

\*************************************************************************
  Setarea orei
*************************************************************************/
void setRTChour(byte hh)
{
   data byte x = hh/10;

   while(RTC_busy());
   XBYTE[regHOUR10] = x;
   XBYTE[regHOUR1] = hh - x*10;
   XBYTE[regD_RTC] = 0x00;
}

\*************************************************************************
  Setarea minutelor
*************************************************************************/
void setRTCmin(byte mm)
{
   data byte x = mm/10;

   while(RTC_busy());
   XBYTE[regMIN10] = x;
   XBYTE[regMIN1] = mm - x*10;
   XBYTE[regD_RTC] = 0x00;
}

\*************************************************************************
  Setarea secundelor
*************************************************************************/
void setRTCsec(byte ss)
{
   data byte x = ss/10;

   while(RTC_busy());
   XBYTE[regSEC10] = x;
   XBYTE[regSEC1] = ss - x*10;
   XBYTE[regD_RTC] = 0x00;
}

\*************************************************************************
  Setarea zilelor
*************************************************************************/
void setRTCday(byte dd)
{
   data byte x = dd/10;

   while(RTC_busy());
   XBYTE[regDAY10] = x;
   XBYTE[regDAY1] = dd - x*10;
   XBYTE[regD_RTC] = 0x00;
}

\*************************************************************************
  Setarea lunii
*************************************************************************/
void setRTCmon(byte val)
{
   data byte x = val/10;

   while(RTC_busy());
   XBYTE[regMON10] = x;
   XBYTE[regMON1] = val - x*10;
   XBYTE[regD_RTC] = 0x00;
}

\*************************************************************************
  Setarea anului (nuami ultimele 2 cifre)
*************************************************************************/
void setRTCyear(byte val)
{
   data byte x = val/10;
```

```
   while(RTC_busy());
   XBYTE[regYEAR10] = x;
   XBYTE[regYEAR1] = val - x*10;
   XBYTE[regD_RTC] = 0x00;
}

\**************************************************************************
  Setarea zilei din saptamana
**************************************************************************/
void setRTCweek(byte val)
{
   while(RTC_busy());
   XBYTE[regWEEK] = val & 0x07;
   XBYTE[regD_RTC] = 0x00;
}
```

2.8. Periferice I²C

Există multe circuite periferice dedicate I²C care sunt fabricate de multe firme. Totuşi. majoritatea acestora sunt produse de Philips, posesoarea mărcii I²C. O listă a acestora se poate găsi la www.semicoductors.philips.com. Câteva dintre cele mai utilizate sunt:

PCF8566: Driver universal pentru afişaj LCD;
PCF8576: Driver universal pentru afişaj LCD;
PCF8570: Memorie SRAM 256x8biţi;
PCF8571: Memorie SRAM 128x8biţi;
PCF8573: Ceas de timp real (RTC);
PCF8583: Ceas de timp real (RTC) şi calendar cu RAM static 256x8biţi;
PCF8574: Opt intrări/ieşiri paralele;
PCF8584: Controler de magistrală I²C;
PCF8591: Convertor 4 canale analog-digital, 1 canal digital-analog pe 8 biţi.

O altă firmă producătoare de periferice I²C este SGS-Thomson (adresă la www.st.com), specializată în special în memorii:

ST24C08: memorie EEPROM 8Ko (4x256x8biţi);
X24645: memorie EEPROM cu protecţie a blocurilor de memorie.

2.8.1 Ceas de timp real

Circuitul integrat Philips PCF8583 este un ceas de timp real şi o memorie I²C. În figura 3.10 este prezentată modul de legare a circuitului PCF8583 la magistrala I²C.

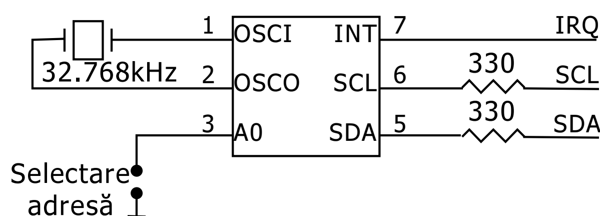


Figura 2.10. Utilizarea circuitului I²C PCF8583

Descrierea circuitului

- tensiunea de alimentare a circuitului: 2,5V÷6V;
- tensiunea de alimentare a ceasului de timp real (0÷70°C): 1.0V÷6V;
- tensiunea necesară pentru menţinerea datelor: 1.0V÷6V;
- curentul consumat în aşteptare (f_{SCL}=0Hz): maxim 50μA;
- data cu calendar pe 4 ani;
- ora în format 12 sau 24;
- baza de timp 32,768Hz sau 50Hz;
- programarea pe interfaţă I^2C;
- adresa I^2C este 1 0 1 0 0 0 A0 R/W;
- incrementarea automată a contorului de adrese;
- programarea alarmei, registrelor de dată/oră;
- generarea de întreruperi;
- RAM CMOS static de 256octeţi;
- oscilator incorporat cu frecvenţa de 32.768kHz;
- autoinţializare la punerea sub tensiune (*Power on Reset*).

Octeţii de la 00h la 07h sunt folosiţi de RTC pentru memorarea stării, datei şi a orei. Octeţii cu adresa între 08h şi 0Fh pot fi folosiţi pentru setarea alarmei sau ca zonă de memorie. Octeţii de la 10h la FFh pot fi folosiţi ca memorie RAM.

PCF8583 poate funcţiona ca ceas (cu tact de 32.768kHz sau de 50Hz) sau ca contor de evenimente, mod în care numărătoarele se vor incrementa pentru fiecare tact al oscilatorului (generatorului de evenimente).

O alarmă poate fi setată la o anumită dată, săptămânală sau zilnică ori la apariţia unui număr de evenimente.

În modul ceas, registrul de la adresa 07h poate fi setat pentru numărarea sutimilor de secundă, secundelor, minutelor sau zilelor. Dacă alarma nu este programată, atunci vor fi contorizate zilele.

Dacă s-a produs un eveniment care a dus la declanşarea alarmei, indicatorul de alarmă din registrul de stare este setat, rămânând setat până când se execută o operaţie de citire.

Structura registrelor şi memoriei interne este prezentată în tabelul 3.15.

<table>
<tr><td colspan="7" align="right">Tabelul 2.17</td></tr>
<tr><td colspan="4" rowspan="2">MOD CEAS
CONTROL/STARE</td><td colspan="2" rowspan="2">MOD CONTOR
CONTROL/STARE</td><td>Adresă</td></tr>
<tr><td>00h</td></tr>
<tr><td colspan="4">Miimi de secundă</td><td rowspan="2">D1</td><td rowspan="2">D0</td><td rowspan="2">01h</td></tr>
<tr><td colspan="3">1/10s</td><td>1/100s</td></tr>
<tr><td colspan="4">Secunde</td><td rowspan="2">D3</td><td rowspan="2">D2</td><td rowspan="2">02h</td></tr>
<tr><td colspan="3">10s</td><td>1s</td></tr>
<tr><td colspan="4">Minute</td><td rowspan="2">D5</td><td rowspan="2">D4</td><td rowspan="2">03h</td></tr>
<tr><td colspan="3">10min</td><td>1min</td></tr>
<tr><td colspan="4">Format, AM/PM, ore</td><td colspan="2" rowspan="2">liber</td><td rowspan="2">04h</td></tr>
<tr><td>F</td><td>A</td><td>10h</td><td>1h</td></tr>
<tr><td colspan="4">An / Zile</td><td colspan="2" rowspan="2">liber</td><td rowspan="2">05h</td></tr>
<tr><td colspan="2">ani</td><td>10z</td><td>1z</td></tr>
<tr><td colspan="4">Ziua / Luni</td><td colspan="2" rowspan="2">liber</td><td rowspan="2">06h</td></tr>
<tr><td colspan="2">ziua</td><td>10l</td><td>1l</td></tr>
<tr><td colspan="4">Timer</td><td colspan="2">Timer</td><td rowspan="2">07h</td></tr>
<tr><td colspan="3">10z</td><td>1z</td><td>T1</td><td>T0</td></tr>
</table>

<table>
<tr><td colspan="4">CONTROL ALARMĂ</td><td colspan="2">CONTROL ALARMĂ</td><td>08h</td></tr>
<tr><td colspan="4">Alarma zecimi de secundă</td><td colspan="2">Alarmă</td><td rowspan="2">09h</td></tr>
<tr><td colspan="3">1/10s</td><td>1/100s</td><td>D1</td><td>D0</td></tr>
<tr><td colspan="4">Alarmă secunde</td><td colspan="2">Alarmă</td><td rowspan="2">0Ah</td></tr>
<tr><td colspan="3">10s</td><td>1s</td><td>D3</td><td>D2</td></tr>
<tr><td colspan="4">Alarmă Minute</td><td colspan="2">Alarmă</td><td rowspan="2">0Bh</td></tr>
<tr><td colspan="3">10m</td><td>1m</td><td>D5</td><td>D4</td></tr>
<tr><td colspan="4">Alarmă ore*</td><td colspan="2" rowspan="2">liber</td><td rowspan="2">0Ch</td></tr>
<tr><td>F</td><td>A</td><td>10h</td><td>1h</td></tr>
<tr><td colspan="4">Alarma an / ziua*</td><td colspan="2" rowspan="2">liber</td><td rowspan="2">0Dh</td></tr>
<tr><td colspan="2">xx</td><td>10z</td><td>1z</td></tr>
<tr><td colspan="4">Alarma ziua / luna</td><td colspan="2" rowspan="2">liber</td><td rowspan="2">0Eh</td></tr>
<tr><td colspan="2">ziua</td><td>10l</td><td>1l</td></tr>
<tr><td colspan="4">Alarmă timer</td><td colspan="2">Alarmă timer</td><td rowspan="2">0Fh</td></tr>
<tr><td colspan="3">10z</td><td>1z</td><td>T1</td><td>T0</td></tr>
<tr><td colspan="4">liber</td><td colspan="2">liber</td><td>10h
⋮
FFh</td></tr>
</table>

în continuare, în tabelul 3.16, sunt caracterizate registrele de programare ale circuitului. Toate registrele interne sunt şterse la iniţializare.

Tabelul 2.18

Bit	Descriere
Registrul de stare şi comandă (adresă 00h)	
0	Indicator de registru contor (dacă bitul de validare a alarmei este 0 atunci acest indicator îşi va schimba starea din secundă în secundă cu un factor de umplere de 50%).
1	Indicator de alarmă (dacă bitul de validare a alarmei este 0 atunci acest indicator îşi va schimba starea din minut în minut cu un factor de umplere de 50%).
2	Bit validare alarmă: 0 → alarmă inhibată. Indicatorii de mai sus vor bascula, spaţiul de memorie de la adresa 08h la 0Fh din RAM este spaţiu liber şi poate fi folosit de utilizator; 1 → alarmă validată (registrul de la adresa 08h este registrul de control al alarmei).
3	Indicator citire mascată: 0 → citirea de la adresele 05h şi 06h se va face nemascat; 1 → citirea directă a contorilor de luni şi zile.
4,5	Mod de funcţionare: 00 → mod ceas 32.768KHz; 01 → mod ceas 50Hz; 10 → mod contor de evenimente; 11 → mod test;
6	Indicator PAUZĂ: 0 → contorizare; 1 → memorarea şi menţinerea ultimelor valori ale contorilor.
7	Indicator STOP: 0 → contorizare; 1 → oprirea şi resetarea contorilor;
Registrul contor ore (adresă 04h)	
0,1 2,3	Orele (unităţile) în format BCD
4,5	Zecile de ore în format binar
6	Indicator AM/PM: 0 = AM, 1= PM
7	Format oră: 0 = format 24h, indicatorul AM/PM nu se modifică; 1 = format 12h, indicatorul AM/PM se va modifica.
Registrul contor ani/zile (adresă 05h)	

0,1, 2,3	Zilele (unităţile) în format BCD
4,5	Zecile de zile (în format binar de la 0 la 3)
6,7	Anii (de la 0 la 3, 0 = an bisect). Dacă bitul citire mascată este setat atunci la citire aceşti biţi vor fi 0.
Registrul contor zi din săptămână/lună (adresă 06h	
Bit	Descriere
0,1 2,3	Luna (unităţile) în format BCD
4	Zecile de luni în format binar de la 0 la 1
5,6 7	Ziua din săptămână, în format binar de la 0 la 6. Dacă bitul citire mascată este setat atunci la citire aceşti biţi vor fi 0.
Registrul control alarmă (adresă 07h)	
Bit	Descriere
0,1 2	Funcţii contor (timer): 000 = fără contor; 001 = sutimi de secundă; 010 = secunde; 011 = minute; 100 = ore; 101 = zile; 110 = nefolosit; 111 = mod test, toţi contorii vor număra în paralel.
3	Validare întreruperi timer: 0 = invalidate, 1 = validate
4,5	Funcţii alarmă: 00 = nu se va declanşa alarma 01 = alarma zilnică (la ora specificată) 10 = alarmă săptămânală (la ziua din săptămână şi ora setate) 11 = alarma la data şi ora setată
6	Validarea alarmei de la contor (timer): 0 = invalidată, 1 = validată
7	Validarea întreruperilor de la alarmă: 0 = invalidate, 1 = validate. Întreruperile vor fi generate numai dacă este setat bitul de validare a întreruperilor din registrul de stare şi control.

Toate registrele de alarmă au aceeaşi structură ca registrele de date şi se găsesc începând cu adresa 08h.

O alarmă este generată atunci când conţinutul registrelor de alarmă au aceeaşi valoare cu registrele de date corespunzătoare (comparare la nivel de bit). Biţii care reprezintă anul şi ziua din săptămână nu vor fi comparaţi la alarma pentru o anumită dată. În cazul alarmei zilnice, nu vor fi luaţi în consideraţie şi biţii care reprezintă luna. În cazul alegeri alarmei săptămânale, conţinutul registrului de alarmare a lunii are funcţiunile din tabelul 3.17.

Tabelul 2.19

Bit	Descriere
0	1 = setare alarmă pentru ziua '0';
1	1 = setare alarmă pentru ziua '1';
2	1 = setare alarmă pentru ziua '2';
3	1 = setare alarmă pentru ziua '3';
4	1 = setare alarmă pentru ziua '4';
5	1 = setare alarmă pentru ziua '5';
6	1 = setare alarmă pentru ziua '6';
7	rezervat.

Ieşire de întrerupere INT

Ieşirea INT (de tip FET-n, cu drenă în gol) este programată prin registrul de control a alarmei. Ieşirea este activă (nivel 0 LOGIC) atunci când indicatorul de alarmă sau indicatorul de contor sunt setaţi. În modul ceas fără alarmă, ieşirea este controlată de indicatorul de contor.

Oscilatorul

Un cristal de 32.768kHz trebuie conectat între OSCI (pinul 1) şi OSCO (pinul 2). Pentru controlul frecvenţei se poate lega un condensator variabil (<100p) între OSCI şi V_{CC}. În modul ceas cu tact extern şi în modul contor de evenimente, oscilatorul intern este inhibat şi OSCI este trecută în starea de înaltă impedanţă, ceea ce permite folosirea unui semnal de 50Hz extern pentru funcţia de ceas sau a unui semnal cu frecvenţă ridicată pentru numărarea evenimentelor.

Iniţializarea

La punerea sub tensiune, interfaţa I^2C, registrele de stare/comandă şi toţi contorii sunt şterşi. Circuitul începe să numere considerând frecvenţa de 32.768kHz, ora în format 24h, 1 ianuarie, anul 00, ora 00:00:00 00. Ieşirea INT va genera un semnal cu frecvenţa de 1Hz, plecând din starea sus. Acest semnal poate fi anulat prin setarea corespunzătoare a registrului de alarmă. Interfaţa I^2C este inhibată şi resetată dacă tensiunea de alimentare scade sub valoarea de funcţionarea a interfeţei.

Este recomandată trecerea în mod inactiv a RTC-ului în timpul setării acestuia cu noile date. Înscrierea de valori eronate în registrele de date conduc la o proastă contorizare dar nu vor bloca funcţionarea circuitului.

Protocolul de legătură I²C

Circuitul RTC PCF8583 se interfaţează prin intermediul unui protocol I²C, protocol descris pe larg în paragraful 1.10, *Interfaţa serială sincronă I²C.* Pentru acest circuit, frecvenţa maximă de lucru pe interfaţa serială este de 100kHz. Sunt prezentate, în figura 3.11, procedurile de lucru cu dispozitivul I²C.

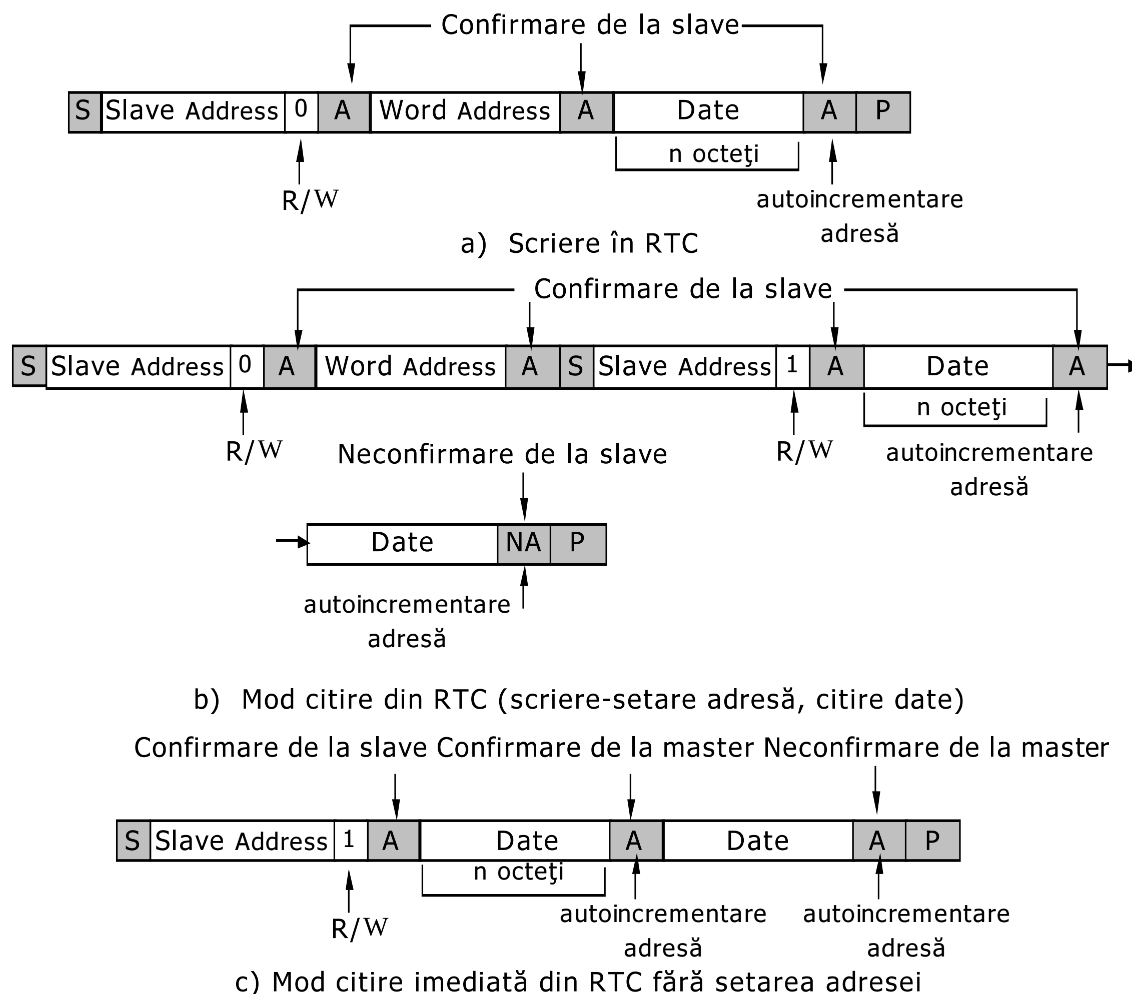


Figura 2.11. Operarea circuitului PCF8583

2.8.2 Convertoare A/D şi D/A

Circuitul integrat PCF8591, fabricat în tehnologie CMOS, conţine 4 intrări analogice, 1 ieşire analogică şi o interfaţă I²C. Adresa I²C se poate stabili folosind trei pini ai circuitului (A0, A1 şi A2), putând fi legate la aceeaşi magistrală I²C opt circuite integrate PCF8591. Adresa circuitului este dată de structura:

1	0	0	1	A2	A1	A0	R/$\overline{W}$

Caracteristicile mai importante ale circuitului sunt:

- tensiune de alimentare între 2,5V şi 6V;
- timp de eşantionare determinat de magistrala I²C;
- patru intrări analogice care pot fi programate în mod comun sau diferenţial;
- autoincrementarea canalelor.

Descrierea pinilor circuitului este prezentată în tabelul 3.20 iar în tabelul 3.21 este prezentă structura octetului de comandă şi control (al doilea octet trimis).

Tabelul 2.20

Pin	Semnal	Descriere
1	AIN0	intrări analogice
2	AIN1	
3	AIN2	
4	AIN3	
5	A0	selectarea adresei I^2C
6	A1	
7	A2	
8	V_{SS}	tensiune de alimentare negativă
9	SDA	linia de date I^2C
10	SCL	linia de tact I^2C
11	OSC	intrare-ieşire oscilator
12	EXT	selecţia oscilatorului intern sau extern
13	AGND	masa analogică
14	V_{REF}	intrare, tensiune de referinţă
15	AOUT	ieşire analogică
16	V_{DD}	tensiune de alimentare pozitivă

Tabelul 2.21

Bit	Descriere
0,1	Numărul canalului A/D: 00 – canalul 0 01 – canalul 1 10 – canalul 2 11 – canalul 3
2	Indicator autoincrementare canal (1 = validat)
3	0
4,5	Programarea intrărilor analogice: AIN0 → channel 0; AIN1 → channel 1; AIN2 → channel 2; AIN3 → channel 3 00: 4 intrări AIN0 (+), AIN3 (–) → channel 0; AIN1 (+), AIN3 (–) → channel 1; AIN2 (+), AIN3 (–) → channel 2 01: 3 intrări diferenţiale AIN0 → channel 0; AIN1 → channel 1; AIN2 (+), AIN3 (–) → channel 2 10: Intrări diferenţiale şi normale combinate AIN0 (+), AIN1 (–) → channel 0; AIN2 (+), AIN3 (–) → channel 1 11: 2 intrări diferenţiale
6	Validare ieşire convertoare D/A (1 = validată)
7	0

MSB							LSB	registru date DAC
D7	D6	D5	D4	D3	D2	D1	D0	

Conversia D/A

Al treilea octet trimis către circuit este memorat în registrul convertorului digital/analog şi este convertit într-o tensiune analogică folosind un divizor rezistiv conectat la V_{REF}.

Valoarea tensiunii de ieşire la pinul AOUT se poate calcula folosind formula:

$$V_{AOUT} = V_{AGND} + \frac{V_{REF} - V_{AGND}}{256} \sum_{i=0}^{7} D_i \times 2^i$$

Conversia A/D

Ciclul de conversie analog/digital începe atunci când este adresat pentru citire (imediat după decodarea unei adrese de citire valide) şi este pornit de frontul descrescător al semnalului SCL atunci când pe SDA se transmite validarea recepţiei (A) valorii conversiei anterioare.

Primul octet trimis întru-un ciclu de citire reprezintă rezultatul conversie din ciclul anterior de citire sau valoarea 80h imediat după punerea sub tensiune.

Oscilatorul

Dacă pinul EXT este conectat la V_{SS} atunci circuitul integrat va folosi oscilatorul intern pentru generarea tactului necesar conversiei A/D. La pinul OSC este disponibil un semnal de frecvenţa oscilatorului (între 0,75MHz şi 1,25MHz).

Dacă pinul EXT este conecta la V_{DD}, pinul OSC este transformat în pin de intrare la care se poate aplica un tact extern care va fi folosit pentru conversia A/D.

2.8.3 Memorii E^2ROM

Memoriile E^2ROM (*electrically erasable ROM*) au căpătat o largă dezvoltare datorită avantajelor oferite de conservarea datelor în lipsa alimentării, coroborat cu simplitatea modificării locaţiilor de memorie.

Memoriile E^2ROM dezvoltate în tehnologie I^2C au capacităţi de până la 8kB (la data culegerii materialului). În continuare este prezentat circuitul ST24C08, o memorie de 1kB împărţită în 4 blocuri de 256 octeţi.

Principalele caracteristici ale memoriei sunt:

- minim 1.000.000 de cicluri de scriere
- minim 10 ani timp de menţinere a datelor înscrise
- tensiune de alimentare între 3V şi 5,5V
- protecţia la scriere a blocurilor
- proces de scriere octet cu octet sau octeţi multipli (până la 8 octeţi)
- proces de scriere a unei întregi pagini (până la 16 octeţi)

- ciclu de scriere autonom
- incrementarea automată a adresei
- protecţie la descărcări electrostatice de până la 4kV (modelul corpului uman).

Funcţionalitatea pinilor capsulei este descrisă în tabelul 3.22, adresa I^2C a memoriei în tabelul 3.23. şi structura octetului de comandă în tabelul 3.24.

Tabelul 2.22

Pin	Semnal	Descriere
1	PRE	Protecţie la scriere
2	NC	
3	E	Selecţie circuit
4	V_{SS}	Masă
5	SDA	Intrare/ieşire date I^2C
6	SCL	Tact I^2C
7	MODE	Mode scriere (0 = mod multioctet, 1 = mod paginat)
8	V_{CC}	Tensiune de alimentare (-0,3V÷6,5V)

Tabelul 2.23

Adresa slave				Selecţie circuit	Selecţie bloc		$R/\overline{W}$
b7	b6	b5	b4	b3	b2	b1	b0
1	0	1	0	E	A9	A8	$R/\overline{W}$

Tabelul 2.24

Operaţia	$R/\overline{W}$	Mode	Octeţi	Condiţie
Citire de la adresa curentă	1	X	1	Start, selecţie circuit, $R/\overline{W}$ = 1
Citire de la o adresă aleatoare	0	X	0	Start, selecţie circuit, adresa, $R/\overline{W}$ = 0
	1	X	1	Restart, adresa, $R/\overline{W}$ = 1
Citire secvenţială	1	X	1÷1024	La fel ca la citirea de la adresa curentă sau aleatoare
Scriere octet	1	X	1	Start, selecţie circuit, $R/\overline{W}$ = 0
Scriere secvenţială	0	1	8	Start, selecţie circuit, $R/\overline{W}$ = 0
Scriere paginată	0	0	16	Start, selecţie circuit, $R/\overline{W}$ = 0

Protecţia la scriere

Blocul superior de 256 de octeţi poate fi protejat la scriere. Protecţia poate începe de la oricare început de pagină de 16 octeţi. Adresa paginii de început a zonei protejate este stabilită de semioctetul superior (b4÷b7) din ultimului octet din memorie (blocul 3 adresa 3FFh), conform tabelului 3.25.

Tabelul 2.25

Adresa 1FFh, bloc 1							
b7	b6	b5	b4	b3	b2	b1	b0
Adresa de început a zonei protejate					0 = Protecţie validată	x	x

Atenţie! În modul de scriere multioctet se poate scrie peste începutul zonei de protecţie dacă adresa primului octet este chiar înainte de începutul zonei protejate.

Adresarea memoriei

Pentru începerea transferului, circuitul master trebuie să transmită un START, adresa I^2C a circuitului, selecţia blocului de memorie şi bitul de direcţie ($R/\overline{W}$).

Operaţia de scriere

Modul de scriere multioctet este validat dacă pinul MODE este legat la V_{DD}, iar dacă este legat la V_{SS} atunci este selectat modul de scriere paginat. Pinul MODE poate fi modificat dinamic. Modul de scriere octet cu octet este independent de valoarea MODE. Dacă pinul MODE este nelegat atunci singurul mod de scriere este cel octet cu octet şi de aceea este recomandată legarea la V_{DD} sau la V_{SS} a acestui pin.

După semnalul START, dispozitivul master transmite adresa circuitului slave terminată cu zero ($R/\overline{W} = 0$) şi astfel dispozitivul slave, după validarea recepţiei, aşteaptă al doilea octet care este interpretat ca adresa locaţiei de memorie din blocul selectat anterior (în primul octet). Circuitul slave va valida recepţia.

Scrierea unui octet

În acest mod, după transmiterea adresei prezentă mai sus, dispozitivul master va transmite un octet care va reprezenta data care va fi înscrisă la adresa selectată. Dispozitivul slave va valida recepţia şi master-ul va transmite STOP. Ciclul de scriere este pornit după primirea semnalului STOP, şi are durata de 10 ms, timp în care memoria nu va răspunde la nici o cerere externă. Procedura de lucru este descrisă în figura 3.12.a).

Scrierea mai multor octeţi

Dacă semnalul MODE este 1 atunci se pot scrie mai mulţi octeţi în aceeaşi operaţie de scriere plecând de la o adresă oarecare. Dispozitivul master poate să transmită până la 8 octeţi care vor fi înscrişi la adrese succesive plecând de la adresa stabilită iniţial. Fiecare octet transmis este validat de circuitul slave iar la sfârşit dispozitivul master trebuie să transmită STOP. Ciclul de scriere este pornit după primirea semnalului STOP. Între două comenzi de scriere (a blocurilor de maxim 8 octeţi) trebuie aşteptat 20 ms pentru ca memoria să definitiveze ciclul de scriere. Procedura de lucru este descrisă în figura 3.12.b).

Scrierea paginată

Acest mod este selectat dacă semnalul MODE este 0. În acest mod se pot scrie în aceeaşi ciclu până la 16 octeţi, cu condiţia ca aceştia să se afle în aceeaşi pagină (biţii din adresă A9÷A4 identici). Ca şi mai sus, dispozitivul master va transmite adresa de început şi apoi octeţii de date. Adresa va fi incrementată automat atunci când circuitul slave validează recepţia octetului. La sfârşit dispozitivul master trebuie sa transmită STOP. Ciclul de scriere este

pornit după primirea semnalului STOP, şi are durata de 10 ms, timp în care memoria nu va răspunde la nici o cerere externă. Procedura de lucru este descrisă în figura 3.12.b).

Citirea din memorie

Operaţia de citire este independentă de valoarea semnalului MODE.

Citirea de la adresa curentă a unuia sau mai mulţi octeţi

Circuitul ST24C08 are un contor de adrese intern care este autoincrementat după fiecare operaţie de scriere sau citire. Contorul se incrementează până la valoarea 1023 (3FFh) după care sare la valoarea 000h. Citirea de la adresa curentă se face prin transmiterea condiţiei de START, adresei dispozitivului slave şi a blocului şi a bitului de direcţie (R/$\overline{W}$ = 1).

Dispozitivul slave va valida recepţia şi va transmite valoarea octetului care se afla la adresa curentă după care o incrementează. Dacă dispozitivul master validează recepţia, dispozitivul slave va transmite următorul octet, proces ca se va repeta până când dispozitivul master nu va mai valida recepţia. În acest moment dispozitivul slave va transmite condiţia de STOP. Procedura de lucru este descrisă în figura 3.12.c).

Pentru început se stabileşte adresa ca la procesul de scriere (primii 2 octeţi după START) după care se transmite un RESTART urmat de adresa circuitului slave, a blocului şi a bitului de direcţie (R/$\overline{W}$ = 1). Astfel, în prima etapă se setează registrul de adrese la valoarea dorită, iar în a doua etapă se citesc datele folosind algoritmul prezentat la citirea de la adresa curentă a unuia sau mai mulţi octeţi. Procedura de lucru este descrisă în figura 3.12.d) şi 3.12.e).

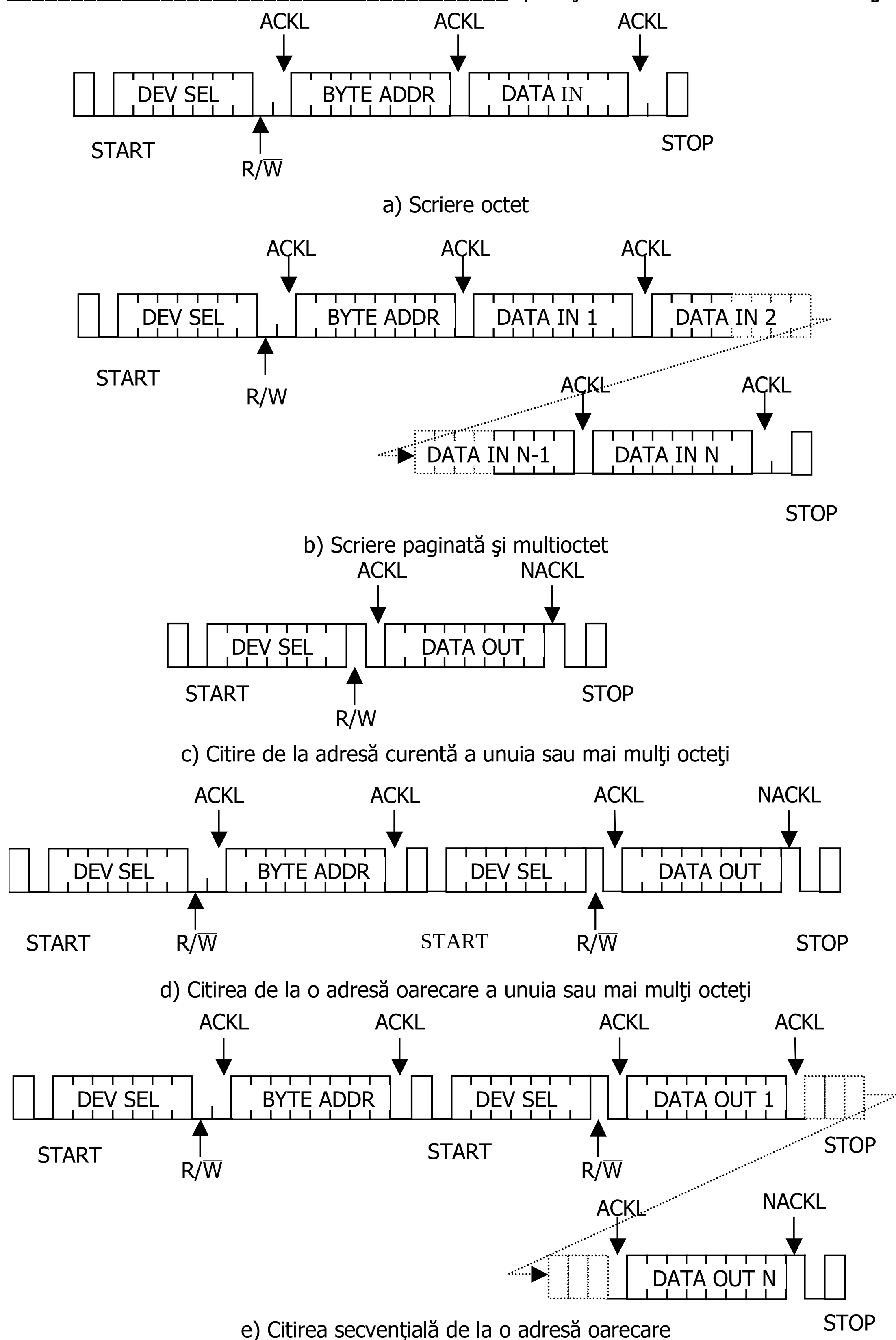


Figura 2.12. Citirea/scrierea datelor în memoria ST24C08

Citirea de la o adresă oarecare a unuia sau mai mulți octeți

2.8.4 Extensii ieşiri paralele

Extensiile paralele reprezintă o alternativă economică de mărire a numărului de intrări/ieşiri digitale a unui sistem cu microcontroler I^2C.

Unul din circuitele care realizează această funcţiune este PCF8574, un dispozitiv cu 16 pini care pot fi programaţi individual ca intrare sau ieşire (ieşire quasi bidirecţionale în sens Intel – pinii de intrare trebuiesc setaţi prin trecere în starea sus). Ieşirile pot comanda direct LED-uri (pinii acceptă 20mA intrare, 25mA ieşire)

Acest circuit generează un semnal de întrerupere ($\overline{INT}$) când semnalul de intrare la oricare pin s-a modificat faţă de ultima citire, ceea ce permite informarea microprocesorului de apariţia unui eveniment fără a fi necesară o comunicaţie pe magistrala I^2C.

Semnificaţia pinilor circuitului este prezentată în tabelul 3.24.

Tabelul 2.26

Pin	Semnal	Descriere
1	A0	Adresa I^2C
2	A1	
3	A2	
4	P0	Pini quasi bidirecţionali
5	P1	
6	P2	
7	P3	
8	VSS	Masa
9	P4	Pini quasi bidirecţionali
10	P5	
11	P6	
12	P7	
13	$\overline{INT}$	Ieşire cu drenă în gol – semnal de întrerupere
14	SCL	Tact I^2C
15	SDA	Intrare/ieşire date I^2C
16	VDD	Tensiune de alimentare pozitivă (2,5V – 6V)

Adresa perifericului I^2C, în variantele PCF8574, respectiv PCF8574A este următoarea:

PCF8574

MSB LSB

0	1	0	0	A2	A1	A0	R/$\overline{W}$

PCF8574A

0	1	1	1	A2	A1	A0	R/$\overline{W}$

Fiecare pin poate fi programat independent ca intrare sau ieşire. Programarea ca intrare se face prin trecerea în starea 1 LOGIC a pinului respectiv. Semnalul de întrerupere este generat atunci când cel puţin unul din pini îşi modifică starea faţă de ultimul acces. Pinul $\overline{INT}$ rămâne zero până la următorul acces la circuit sau până când starea semnalelor de la pinii P0÷P7 coincide cu starea memorată.

După iniţializare, toate ieşirile sunt în starea 1 LOGIC putând fi folosite ca intrări.

2.8.5 Emularea unei interfeţe I^2C

Anumite aplicaţii necesită interfaţarea la o magistrală I^2C şi a unor circuite care nu au această facilitate. Există posibilitatea simulării interfeţei I^2C pe un port paralel de PC, pe o interfaţă serială asincronă etc.

În cele ce urmează este prezentat, în asamblor, programul sursă pentru simularea interfeţei I^2C pe un microcontroler din familia 8xC51.

```
/***********************************************************************\
**  Descriere:    Rutina simulare interfaţa IIC                       **
**                                                                     **
**  Versiune:     1.0                                                  **
**  Inceut la:    Iunie 94                                             **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania   **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                   **
**                                                                     **
**  Acest modul contine urmatoarele functii:                           **
**                                                                     **
**  Ultima modificare la data: 23 nov 1998                             **
**  Istoric:                                                           **
**                                                                     **
**  Observatii:                                                        **
**                                                                     **
\***********************************************************************/

$REGISTERBANK (0,1)
NAME    IIC
$NOLIST
; Etichete PUBLIC:
; ************************************
        public  ?IIC_Test_Device
        public  ?IIC_Test_Device?BYTE
        public  ?IIC_Write
        public  ?IIC_Write?BYTE
        public  ?IIC_Write_Sub
        public  ?IIC_Write_Sub?BYTE
        public  ?IIC_Write_Sub_SWInc
        public  ?IIC_Write_Sub_SWInc?BYTE
        public  ?IIC_Write_Memory
        public  ?IIC_Write_Memory?BYTE
        public  ?IIC_Write_Sub_Write
        public  ?IIC_Write_Sub_Write?BYTE
        public  ?IIC_Write_Sub_Read
        public  ?IIC_Write_Sub_Read?BYTE
        public  ?IIC_Read
        public  ?IIC_Read?BYTE
        public  ?IIC_Read_Sub
        public  ?IIC_Read_Sub?BYTE
        public  ?IIC_Read_Status
        public  ?IIC_Read_Status?BYTE
        public  ?Init_IIC
        public  ?Init_IIC?BYTE
;
;       *****************
;       * Constante IIC *
;       *****************
IICRetries      equ 5           ;Numar incercari
IICTimeOut      equ 100         ;100 ms Time Out IIC
NVM_Write_Delay equ 40          ;40 ms acces EEPROM
;
;       *******************************************************************
;       * Constante control IICInOut                                      *
;       * Semioctetul LOW: stare; semioctetul HIGH: control               *
;       *******************************************************************
;
SLREAD          equ 7                   ;Citeste Slave = 1
mSlread         equ 1 SHL SLREAD        ;
SLNOMEM         equ 6                   ;Memorie Slave inexistenta = 1
mSlnomem        equ 1 SHL SLNOMEM       ;
SLNOSUB         equ 5                   ;Slave subadresa inexistenta = 1
mSlnosub        equ 1 SHL SLNOSUB       ;
SLNOINCR        equ 4                   ;Autoincrementare subadresa = 1
mSlnoincr       equ 1 SHL SLNOINCR      ;
_Test_Device     equ mSlnomem+mSlnosub          ;Test dispozitiv
_Write           equ mSlnomem+mSlnosub          ;Scrie fara subadresa
_Write_Sub       equ mSlnomem                   ;Scrie cu subadrresa si autoinc.
_Write_Memory    equ mSlnoincr                  ;Scrie memorie (octet cu octet+intarziere)
_Write_Sub_SWInc equ mSlnomem+mSlnoincr         ;Scrie cu subadrresa fara autoinc.
_Read_Sub        equ mSlnomem+mSlread           ;Citeste cu subadresa
```

```
_Read            equ mSlnomem+mSlnosub+mSlread  ;Citeste fara subadresa
;
ERROR           equ 0                    ;Eroare generala
mError          equ 1 SHL ERROR          ;
MSTNOTREADY     equ 1                    ;Master \Ready
mMstNotReady    equ 1 SHL MSTNOTREADY    ;
BYTE1EXPECTED   equ 2                    ;Asteapta primul bit
mByte1Expected  equ 1 SHL BYTE1EXPECTED  ;
INBLOCK2        equ 3                    ;E/R Bit 2
mInBlock2       equ 1 SHL INBLOCK2       ;
;       *******************************************************
;       * Data / Definitii registre                           *
;       *******************************************************
;
;
IICBit          Segment Bit
                Rseg    IICBit
;
                public  IIC_Error
IIC_Error:      dbit    1               ;Bit Eroare IIC
;
IICBitAdr       Segment Data BitAddressable
                Rseg    IICBitAdr
;
IICCntrl:       ds      1               ;Registru control bit-addesabil IIC
;
IICPar          Segment Data
                Rseg    IICPar
;
SlaveAddress:   ds      1               ;Adresa Slave + R/W
DataCount1:     ds      1               ;Contor bloc 1
DataIndex1:     ds      1               ;Adresa buffer transfer block 1
SubAddress:     ds      1               ;Subadresa (daca exista)
DataCount2:     ds      1               ;Block 2 Count
DataIndex2:     ds      1               ;Block 2 Transfer Buffer Address
;
;       *****************************************************************
;       *  Simboluri utilizte in proceduri                              *
;       *                                                               *
;       *       Slv             - Adresa slave                          *
;       *       SlvW            - Adresa slave + Write                  *
;       *       SlvR            - Adresa slave + Read                   *
;       *       Sub             - Subadresa                             *
;       *       D1[0..L-1]      - Sir date (DataIndex1)                 *
;       *       L               - Lungime D1 (DataCount1)               *
;       *       D2[0..M-1]      - Sir date (DataIndex2)                 *
;       *       M               - Lungime D2 (DataCount2)               *
;       *       S               - Start                                 *
;       *       P               - Stop                                  *
;       *       A               - Ack                                   *
;       *       N               - Nack                                  *
;       *                                                               *
;       *****************************************************************
IICCode         Segment Code InBlock
                Rseg    IICCode
;
; ******************************************************************************
; * IIC_Test_Device:                                                           *
; *     PROCEDURE ( SlaveAddress ) BYTE/BIT/None ;                             *
; *                                                                            *
; * Format IIC:S SlvW A P                                                      *
; *                                                                            *
; ******************************************************************************
?IIC_Test_Device:
                mov     DataCount1,#0           ;\Data
                mov     a,#_Test_Device
                sjmp    IICInOutBlock1
;
; ******************************************************************************
; * IIC_Write_Memory:                                                          *
; *     PROCEDURE ( SlaveAddress, Count, SourcePtr, SubAddr ) BYTE / BIT / None *
; *                                                                            *
; *     L   = Contor                                                           *
; *     D1  = Pointer sursa                                                    *
; *     Sub = Subadresa                                                        *
; *                                                                            *
; * Format IIC:S SlvW A Sub     A D1[0]   A P  { DelayNVMemory }               *
; *             S SlvW A Sub+1   A D1[1]   A P  { DelayNVMemory }              *
; *             ................................................               *
; *             S SlvW A Sub+L-1 A D1[L-1] A P  { DelayNVMemory }              *
; *                                                                            *
; ******************************************************************************
?IIC_Write_Memory:
                mov     a,#_Write_Memory
```

```
                sjmp    IICInOutBlock1
;
; ****************************************************************************
; * IIC_Read_Sub:                                                            *
; *     PROCEDURE ( SlaveAddress, Count, DestPtr, SubAddr ) BYTE / BIT / None ; *
; *                                                                          *
; *     M   = Contor                                                         *
; *     D2  = Pointer destinatie                                             *
; *     Sub = Subadresa                                                      *
; *                                                                          *
; * Format IIC:S SlvW A Sub A S SlvR A D2[0] A D2[1] A ..... A D2[L-1] N P      *
; *                                                                          *
; ****************************************************************************
?IIC_Read_Sub:   mov     a,#_Read_Sub
                 sjmp    IICInOutBlock2
;
; ****************************************************************************
; * IIC_Write_Sub:                                                           *
; *     PROCEDURE ( SlaveAddress, Count, SourcePtr, SubAddr ) BYTE / BIT / None *
; *                                                                          *
; *     L   = Contor                                                         *
; *     D1  = Pointer sursa                                                  *
; *     Sub = Subadresa                                                      *
; *                                                                          *
; * Format IIC:S SlvW A Sub A D1[0] A D1[1] A ..... A D1[L-1] A P            *
; *                                                                          *
; ****************************************************************************
?IIC_Write_Sub:
                mov     a,#_Write_Sub
                sjmp    IICInOutBlock1
;
; ****************************************************************************
; * IIC_Write_Sub_SWInc:                                                     *
; *     PROCEDURE ( SlaveAddress, Count, SourcePtr, SubAddr ) BYTE / BIT / None *
; *                                                                          *
; *     L = Contor                                                           *
; *     D1= Pointer sursa                                                    *
; *     Sub = Subadresa                                                      *
; *                                                                          *
; * Format IIC:S SlvW A Sub     A D1[0]   A P                                *
; *              S SlvW A Sub+1   A D1[1]   A P                              *
; *              ..............................                              *
; *              S SlvW A Sub+L-1 A D1[L-1] A P                              *
; *                                                                          *
; ****************************************************************************
?IIC_Write_Sub_SWInc:
                mov     a,#_Write_Sub_SWinc
                sjmp    IICInOutBlock1
;
; ****************************************************************************
; * IIC_Write:                                                               *
; *     PROCEDURE ( SlaveAddress, Count, SourcePtr ) BYTE / BIT / None ;     *
; *                                                                          *
; *     L = Contor                                                           *
; *     D1= Pointer sursa                                                    *
; *                                                                          *
; * Format IIC: S SlvW A D1[0] A D1[1] A ..... A D1[L-1] A P                 *
; *                                                                          *
; ****************************************************************************
?IIC_Write:      mov     a,#_Write
                 sjmp    IICInOutBlock1
;
; ****************************************************************************
; * IIC_Read_Status:                                                         *
; *     PROCEDURE ( SlaveAddress, DestPtr ) BYTE / BIT / None ;              *
; *                                                                          *
; *     M = 1                                                                *
; *     D2= Pointer destinatie                                             *
; *                                                                          *
; * Format IIC:S SlvR A D2[0] N P                                            *
; *                                                                          *
; ****************************************************************************
?IIC_Read_Status:
                mov     DataIndex1,DataCount1
                mov     DataCount1,#1
                ; Urmeaza IIC_Read
;
; ****************************************************************************
; * IIC_Read:                                                                *
; *     PROCEDURE ( SlaveAddress, Count, DestPtr ) BYTE / BIT / None ;       *
; *                                                                          *
; *     M = Contor                                                           *
; *     D2= Pointer destinatie                                             *
```

```
; *                                                                           *
; * Format IIC: S SlvR A D2[0] A D2[1] A ..... A D2[M-1] N P                  *
; *                                                                           *
; *****************************************************************************
?IIC_Read:
                orl     SlaveAddress,#1 ;Set bit citire in adresa slave
                mov     a,#_Read
                sjmp    IICInOutBlock2
;
; *****************************************************************************
; * IIC_Write_Sub_Write:                                                      *
; *     PROCEDURE ( SlaveAddress, Count1, SourcePtr1, SubAddr,                *
; *                             Count2, SourcePtr2 ) BYTE / BIT / None ;      *
; *                                                                           *
; *     L   = Contor1                                                         *
; *     D1  = Pointer sursa 1                                                 *
; *     M   = Contor 2                                                        *
; *     D2  = Pointer sursa 2                                                 *
; *     Sub = Subadresa                                                       *
; *                                                                           *
; * Format IIC:S SlvW A Sub A D1[0] A D1[1] A ..... A D1[L-1] A               *
; *                         D2[0] A D2[1] A ..... A D2[M-1] A P               *
; *                                                                           *
; *****************************************************************************
?IIC_Write_Sub_Write:
                mov     a,#_Write_Sub
                sjmp    _IICInOut
; *****************************************************************************
; * IIC_Write_Sub_Read:                                                       *
; *     PROCEDURE ( SlaveAddress, Count1, SourcePtr1, SubAddr,                *
; *                             Count2, DestPtr2 ) BYTE / BIT / None ;        *
; *                                                                           *
; *     L   = Contor1                                                         *
; *     D1  = Pointer sursa 1                                                 *
; *     M   = Contor 2                                                        *
; *     D2  = Pointer destinatie 2                                            *
; *     Sub = Subadresa                                                       *
; *                                                                           *
; * Format IIC:S SlvW A Sub A D1[0] A D1[1] A ..... A D1[L-1] A S SlvR A      *
; *                         D2[0] A D2[1] A ..... A D2[M-1] N P               *
; *                                                                           *
; *****************************************************************************
?IIC_Write_Sub_Read:
                mov     a,#_Read_Sub
                sjmp    _IICInOut
;
IICInOutBlock2: mov     DataCount2,DataCount1
                mov     DataIndex2,DataIndex1
                mov     DataCount1,#0
                sjmp    _IICInOut
;
IICInOutBlock1: mov     DataCount2,#0
                ; Fall Into _IICInOut
;
;       ***********************************************************************
;       *  _IICInOut                                                          *
;       *                                                                     *
;       *       Intrari: definitie octet control din acumulator               *
;       *                                                                     *
;       *       - Bit 7 = 1     Citeste slave                                 *
;       *               = 0     Scrie slave                                   *
;       *       - Bit 6 = 1     Slave nu este EEPROM                          *
;       *               = 0     Slave este EEPROM                             *
;       *       - Bit 5 = 1     Emisie fara subadresa                         *
;       *               = 0     Emisie cu subadresa                           *
;       *       - Bit 4 = 1     Slave fara autoinc. subadresa                 *
;       *               = 0     Slave cu autoinc. subadresa                   *
;       *       - Bit 3..Bit 0  0                                             *
;       *                                                                     *
;       *       Iesire:                                                       *
;       *       Stare daca transmisia a fost reusita:                         *
;       *                       A =  0, Carry = 0, IIC_Error = 0              *
;       *                   daca au fost erori la transmisie                  *
;       *                       A <> 0, Carry = 1, IIC_Error = 1              *
;       *                                                                     *
;       ***********************************************************************
_IICInOut:      anl     IICCntrl,#7             ;Stergere biti control
                orl     IICCntrl,a              ;Setare biti control
;
                jb      IICCntrl.SLNOSUB,IICInOutKernel  ;Jump daca \subadresa
                jnb     IICCntrl.SLNOINCR,IICInOutKernel ;Jump daca autoinc. subadr.
;
;               ************************************************
```

```
;               * Slave cu \autoinc. subadresa                  *
;               *************************************************
                mov     DataIndex2,DataCount1           ;Foloseste DataIndex2 ca nr. local
                mov     DataCount1,#1                   ;Seteaza contor bloc 1 la 1
IICNextByte:    acall   IICInOutKernel                  ;Emisie un octet
                jnz     IICExit                         ;Eroare
                inc     SubAddress                      ;Incrementare subadresa
                inc     DataIndex1                      ;Incrementare DataIndex1
                jb      IICCntrl.SLNOMEM,IICSkipDelay   ;Omite daca \EEPROM
                jb      IICCntrl.SLREAD,IICSkipDelay    ;Omite daca \scriere
;
;               **********************************
;               * Intarziere dupa scriere EEPROM *
;               **********************************
                mov     r1,#2*NVM_Write_Delay           ;intarziere 2*NVM_Write_Delay*0.5s
IICNVM_Delay1:  mov     r0,#249                         ;intariere 0.5 ms
IICNVM_Delay2:  djnz    r0,IICNVM_Delay2                ;
                djnz    r1,IICNVM_Delay1
;
IICSkipDelay:   djnz    DataIndex2,IICNextByte          ;Octet urmator
                sjmp    IICExitOk                       ;Eroare
;
;       ***********************************************************************
;       * IICInOutKernel                                                      *
;       ***********************************************************************
;
IICInOutKernel: mov     r2,#IICRetries                       ;Revenire eroare ( Norm. 5 )
IICInOutRetry:
                clr     ea                                   ;dezactivare intreruperi
                setb    sta                                  ;Mod transfer master
                anl     IICCntrl,#NOT (mError+mInBlock2)     ;Sterge eroare, Start cu Block 1
                setb    IICCntrl.MSTNOTREADY                 ;Master \Ready
                setb    ea                                   ;Validare intreruperi
;
;               ***********************************************
;               * Asteptare terminare mesaj IIC, cu TimeOut   *
;               ***********************************************
                mov     r1,#IICTimeOut                  ;IICTimeOut*1 ms
IICWaitReady1:  mov     r0,#250                         ;Bucla=1 ms
IICWaitReady2:  jnb     IICCntrl.MSTNOTREADY,IICReady   ;Bit este 0 daca este gata
                djnz    r0,IICWaitReady2
                djnz    r1,IICWaitReady1
;
;               ***********************************************
;               * A survenit timeout; eliberare magistrala    *
;               ***********************************************
                clr     ea                              ;dezactivare intreruperi
                mov     a,#mError                       ;modificare stare
                mov     r0,#IICCntrl                    ;
                xchd    a,@r0                           ;
                mov     s1con,#NSTA_STO_AA              ;forteaza eliberare magistrala
                setb    ea                              ;validare intreruperi
;
;               ****************************************
;               * Master a transferat; test eroare     *
;               ****************************************
IICReady:       jnb     IICCntrl.ERROR,IICExitOk        ;Daca \Error intoarce OK
                djnz    r2,IICInOutRetry                ;Repeta pana la nr. maxim incercari
IICErrorExit:   mov     a,#1                            ;intoarce eroare
                sjmp    IICExit                         ;
IICExitOk:      clr     a                               ;Intoarce OK
IICExit:        mov     c,acc.0                         ;Eroare in Carry
                mov     IIC_Error,c                     ;Eroare in IIC_Error
                ret
IICInitCode     Segment Code Unit
                Rseg    IICInitCode
;
;       *****************************************************************
;       * Init_IIC:                                                     *
;       *    PROCEDURE ( OwnSlaveAddress ) ;                            *
;       *    DECLARE OwnSlaveAddress BYTE ;                             *
;       *                                                               *
;       *    Initializeaza IIC SIO1, seteaza adresa slave propriu       *
;       *                                                               *
;       *    NOTA: Aceasta procedura trebuie apelata inaintea apelului  *
;       *          oricearei alte proceduri IIC                         *
;       *****************************************************************
?Init_IIC:      mov     s1adr,SlaveAddress              ;Seteaza adresa slave propriu
                mov     s1con,#NSTA_NSTO_AA             ;Declara Ack in adresa slave
                mov     IICCntrl,#0                     ;Initializare registru ctrl/stare
                setb    es1                             ;Validare intrerupere SIO1
;
;       ***********************************************************
```

```
;       * Modulul este folosit pt setare viteza IIC de timer T1   *
;       **********************************************************
;               orl     tmod,#00100000b                 ;Set T1 AutoReload (Mode 2)
;               mov     th1,#IICTimerReload             ;Setare registre T1
;               mov     tl1,#IICTimerReload             ;
;               setb    tr1                             ;Start timer
;
                ljmp    Init_Slave
;       *****************************************
;       * Vector intrerupere absolut SIO1       *
;       *****************************************
                Cseg    At 002Bh                        ;Vector intrerupere SIO1
                push    psw
                mov     psw,#08h                        ;selectare RegisterBank 1
                ljmp    IIC_Int
;
IICIntCode      Segment Code Unit
                Rseg    IICIntCode
;
;       *****************************************
;       * IIC_int                               *
;       *   Intrerupere SIO1                    *
;       *****************************************
IIC_Int:        push    acc             ;Salvare acumulator
                push    s1sta           ;Salvare S1STA
                mov     a,#High SIOStates ;Transfera controlul la stari
                push    acc
                ret
IICIntStates    Segment Code Page
                Rseg    IICIntStates
;       *********************************************************
;       *       Intrerupere IIC                                 *
;       *                                                       *
;       *       Trebuie sa inceapa la pagina noua               *
;       *       Fiecare rutina de start trebuie sa fie de 8 octeti      *
;       *       Pt mai mult de 8 octeti se poate face salt      *
;       *                                                       *
;       *       Mod transmisie master:                          *
;       *       S SlvW   A Sub    A +                           *
;       *       + D1[0]  A D1[1]  A .....  A D1[L-1]  A +       *
;       *       + D2[0]  A D2[1]  A .....  A D2[M-1]  A P       *
;       *       (Nota: Subadresa este optionala, L si M pot fi 0)       *
;       *                                                       *
;       *       Mod E/R master:                                 *
;       *       S SlvR   A D2[0]  A .....  A D2[M-1]  N P       *
;       *       (Nota: M > 0)                                   *
;       *                                                       *
;       *       S SlvW   A Sub    A +                           *
;       *       + D1[0]  A .....  A D1[L-1]  A S SlvR A +       *
;       *       + D2[0]  A .....  A D2[M-1]  N P                *
;       *                                                       *
;       *********************************************************
;
SIOStates:
;               *********************************************************
;               * Stare 00:                                             *
;               *       eroare bus mod master sau slave datorita unui   *
;               *       START sau STOP eronat                           *
;               * Actiuni                                               *
;               *       IICCntrl := Eroare                              *
;               *       Eliberare magistrala                            *
;               *       Mod slave neadresat                             *
;               *********************************************************
                acall   SetError                ;Actualizare stare
                mov     s1con,#NSTA_STO_AA      ;Eliberare bus, Mod slave neadresat
                ajmp    SIO1Exit                ;Exit
                ds      1                       ;2 + 3 + 2 + 1
;
;               *********************************************************
;               * Stare 08:     (mod master transmisie)                 *
;               *       A fost transmis START                           *
;               * Actiuni                                               *
;               *       Initializare DataCount (r2) si DataPointer (r1) *
;               *       pt. DataBlock1                                  *
;               *       Emisie SLA + R/W, ACK va fi receptionat         *
;               *********************************************************
                mov     r2,DataCount1   ;Initializare DataCount (r2)
                mov     r1,DataIndex1   ;Initializare DataPointer (r1)
                clr     a               ;Implicit: scriere
                ajmp    SendSlvAdr      ;Comun cu starea 10
                ds      1               ;2 + 2 + 1 + 2 + 1
;
;               *********************************************************
```

```
;               * Stare 10:     (mod master transmisie)                 *
;               *       A fost transmis START  repetat                  *
;               * Actiuni                                               *
;               *       Emisie SLA + R, ACK va fi receptionat           *
;               *********************************************************
                mov     a,#1                    ;Set bit citire
SendSlvAdr:     orl     a,SlaveAddress          ;Combinare cu adresa slave
                clr     sta                     ;Sterge bit start
                ajmp    SendAccExit             ;Emisie date, sterge SI
;
;               *********************************************************
;               * Stare 18:     (mod master transmisie)                 *
;               *       A fost transmis SLA+W, ACK a fost receptionat   *
;               * Actiuni                                               *
;               *       If SUBADDRESS Then                              *
;               *         Emisie subadresa, ACK va fi receptionat       *
;               *       Else                                            *
;               *         Emisie date sau STOP                          *
;               *********************************************************
                jb      IICCntrl.SLNOSUB,JumpSendData ;Emisie date daca \subadresa
                mov     a,SubAddress            ;Incarca subadresa
                ajmp    SendAccExit             ;Emisie
                ds      1                       ;3 + 2 + 2 + 1
;
;               *********************************************************
;               * Stare 20:     (mod master transmisie)                 *
;               *       A fost transmis SLA+W, NACK a fost receptionat  *
;               * Actiuni                                               *
;               *       IICCntrl := Eroare                              *
;               *       Emisie STOP                                     *
;               *********************************************************
State20:        acall   SetError                ;Intoarce stare eroare
                ajmp    SendStop                ;Emisie STOP
                ds      4                       ;2 + 2 + 4
;
;               *********************************************************
;               * Stare 28:     (mod master transmisie)                 *
;               *       A fost transmis DATA din S1DAT, ACK a fost rec. *
;               * Actiuni                                               *
;               *       Emisie date                                     *
;               *********************************************************
JumpSendData:   ajmp    SendData                ;Emisie date
;
MstRecHandleAck:cjne    r2,#1,KeepAA            ;Daca <> 1 atunci AA=1
                clr     aa                      ;Altfel NACK
KeepAA:         ret                             ;2 + 3 + 2 + 1
;
;               *********************************************************
;               * Stare 30:     (mod master transmisie)                 *
;               *       A fost transmis DATA din S1DAT, a fost rec.NACK *
;               * Actiuni                                               *
;               *       IICCntrl := Eroare                              *
;               *       Emisie STOP                                     *
;               *********************************************************
                ajmp    State20                 ;La fel cu starea 20
;
SetError:       mov     r0,#IICCntrl            ;Modifica numai starea
                mov     a,#mError               ;Setare indicator eroare
                xchd    a,@r0                   ;
                ret                             ;2 + 2 + 2 + 1 + 1
;
;               *********************************************************
;               * Stare 38:     (mod master transmisie)                 *
;               *       Arbitrare pierduta in SLA, R/W sau DATA         *
;               * Stare 38:     (mod master receptie)                   *
;               *       Arbitrare pierduta la returnare ACK             *
;               * Actiuni                                               *
;               *       Set STA pt Start mod master cand busul e liber  *
;               *       Set AA (AA era 0)                               *
;               *********************************************************
                mov     s1con,#STA_NSTO_AA      ;Setare START si declarare ACK
                clr     IICCntrl.INBLOCK2       ;Revenire la bloc 1
                ajmp    ClrSiExit               ;Stergere SI
                ds      1                       ;3 + 2 + 2 + 1
;
;               *********************************************************
;               * Stare 40:     (mod master receptie)                   *
;               *       A fost transmis SLA+R, a fost receptionat ACK   *
;               * Actiuni                                               *
;               *       Setare DataCount si DataIndex pt bloc 2         *
;               *       If DataCount <> 1 Then                          *
;               *         ACK                                           *
;               *       Else                                            *
```

```
;               *         NACK                                      *
;               *****************************************************
                acall   SetupBlock2             ;Setare bloc 2
                acall   MstRecHandleAck         ;Manevrare Ack
                ajmp    ClrSiExit               ;Exit
                ds      2                       ;2 + 2 + 2 + 2
;
;               *****************************************************
;               * Stare 48:     (mod master receptie)               *
;               *       A fost transmis SLA+R, a fost receptionat NACK  *
;               * Actiuni                                           *
;               *       IICCntrl := Eroare                          *
;               *       Emisie STOP                                 *
;               *****************************************************
                ajmp    State20                 ;Ca la starea 20
;
FetchData:      dec     r2                      ;Decrementare contor date
                mov     @r1,s1dat               ;Slavare date rec.
                inc     r1                      ;Actualizare pointer date
                ret
                ds      1                       ;2 + 1 + 2 + 1 + 1 + 1
;
;               *****************************************************
;               * Stare 50:     (mod master receptie)               *
;               *       A fost rec DATA, a fost returnat ACK        *
;               * Actiuni                                           *
;               *       Aduce data, decr. contor, incr. pointer     *
;               *       If Count = 1 Then                           *
;               *         NACK                                      *
;               *       Else                                        *
;               *         ACK                                       *
;               *****************************************************
                acall   FetchData       ;Data
                acall   MstRecHandleAck ;Ack
                ajmp    ClrSiExit       ;Exit
                ds      2               ;2 + 2 + 2 + 2
;
;               *****************************************************
;               * Stare 58:     (mod master receptie)               *
;               *       A fost rec DATA, a fost returnat NACK       *
;               * Actiuni                                           *
;               *       Set AA                                      *
;               *       Emisie STOP                                 *
;               *****************************************************
                acall   FetchData       ;Data
                setb    aa              ;ACK
                ajmp    SendStop        ;Emisie STOP
                ds      2               ;2 + 2 + 2 + 2
;
;               *****************************************************
;               * Stare 60:     (mod slave receptie)                *
;               *       A fost rec SLA+W, a fost returnat ACK       *
;               * Actiuni                                           *
;               *       IICCntrl (Asteapta octet 1 )                *
;               *****************************************************
State60:        setb    IICCntrl.BYTE1EXPECTED  ;Actualizare stare
                ajmp    ClrSiExit               ;Exit
;
                ds      4                       ;2 + 2 + 4
;
;               *****************************************************
;               * Stare 68     (mod slave receptie)                 *
;               *       Arbitrare pierduta in SLA, R/W ca master.   *
;               *       Receptionata SLA+W propriu, A fost returnat ACK *
;               * Actiuni                                           *
;               *       IICCntrl (Asteapta octet 1 )                *
;               *       Set STA si AA                               *
;               *****************************************************
State68:        setb    IICCntrl.BYTE1EXPECTED  ;Actualizare stare
                clr     IICCntrl.INBLOCK2       ;Reset stare master
                setb    aa                      ;ACK
                ajmp    SendStart               ;Set bit start
;
;               *****************************************************
;               * Stare 70     (mod slave receptie)                 *
;               *       Receptionat apel CALL+W. A fost returnat ACK    *
;               *       (daca apelul general este validat)          *
;               * Actiuni                                           *
;               *       Ca la starea 60                             *
;               *****************************************************
                ajmp    State60                 ;Ca la starea 60
                ds      6
;
```

```
;               *****************************************************
;               * Stare 78     (mod slave receptie)                 *
;               *       Arbitrare pierduta ca SLA, R/W ca master.   *
;               *       Receptionat apel CALL+W. A fost returnat ACK *
;               *       (daca apelul general este validat)          *
;               * Actiuni                                           *
;               *       Ca la starea 68                             *
;               *****************************************************
                ajmp    State68                 ;Ca la starea 68
                ds      6
;
;               *****************************************************
;               * Stare 80     (mod slave receptie)                 *
;               *       Adresat anterior cu SLA propriu. Octeti rec. *
;               *       ACK returnat                                *
;               * Actiuni                                           *
;               *       Receptie slave                              *
;               *****************************************************
State80:        acall   Receive_Slave           ;Receptie slave
                ajmp    ClrSiExit               ;Exit
                ds      4                       ;2 + 2 + 4
;
;               *****************************************************
;               * Stare 88     (mod slave receptie)                 *
;               *       Adresat anterior cu SLA propriu. Octeti rec. *
;               *       NACK returnat (neadresat ca slave in continuare)*
;               * Actiuni                                           *
;               *       Citeste date false, ACK                     *
;               *****************************************************
State88:        mov     a,s1dat                 ;Date dummy
                setb    aa                      ;ACK
                ajmp    ClrSiExit               ;Exit
                ds      2                       ;2 + 2 + 2 + 2
;
;               *****************************************************
;               * Stare 90     (mod slave receptie)                 *
;               *       Adresat anterior cu apel general. Octeti rec. *
;               *       ACK returnat                                *
;               * Actiuni                                           *
;               *       Ca la starea 80                             *
;               *****************************************************
                ajmp    State80                 ;Ca la starea 80
;
SetupBlock2:    setb    IICCntrl.INBLOCK2       ;bloc2
                mov     r1,DataIndex2           ;Pointer la bloc2
                ajmp    SetupB2Proceed          ;
                                                ;2 + 2 + 2 + 2
;
;               *****************************************************
;               * Stare 98     (mod slave receptie)                 *
;               *       Adresat anterior cu apel general. Octeti rec. *
;               *       NACK returnat                               *
;               * Actiuni                                           *
;               *       Ca la starea 88                             *
;               *****************************************************
                ajmp    State88                 ;Ca la starea 88
;
SetupB2Proceed: mov     r2,DataCount2           ;Lungime bloc2
                ret
                ds      3                       ;2 + 2 + 1 + 3
;
;               *****************************************************
;               * Stare A0     (mod slave receptie)                 *
;               *       S-a receptionat STOP sau apel general cand era *
;               *       adresat ca slave rec sau emisie             *
;               * Actiuni                                           *
;               *       Clear IICCntrl (asteapta octet 1)           *
;               *****************************************************
StateA0:        clr     IICCntrl.BYTE1EXPECTED  ;Actualizare stare
                setb    aa                      ;Ack adresa proprie slave
                ajmp    ClrSiExit               ;Exit
                ds      2                       ;2 + 2 + 2 + 2
;
;               *****************************************************
;               * Stare A8     (mod slave emisie)                   *
;               *       S-a receptionat SLA+R. S-a returnat ACK     *
;               *       adresat ca slave rec sau emisie             *
;               * Actiuni                                           *
;               *       Emisie slave                                *
;               *****************************************************
StateA8:        acall   Send_Slave                ;Emisie date slave
                ajmp    ClrSiExit                 ;Exit
                ds      4                         ;2 + 2 + 4
```

```
;
;               ********************************************************
;               * Stare B0    (mod slave emisie)                       *
;               *       Arbitrare pierduta in SLA, R/W ca master.      *
;               *       S-a receptionat SLA+W propriu. ACK returnat    *
;               * Actiuni                                              *
;               *       Set STA + AA                                   *
;               *       Ca la starea A8                                *
;               ********************************************************
                setb    sta                     ;Set bit start
                setb    aa                      ;Set ACK
                ajmp    StateA8                 ;Emisie date
                ds      2                       ;2 + 2 + 4
;
;               ********************************************************
;               * Stare B8    (mod slave emisie)                       *
;               *       S-au transmis datele. ACK returnat             *
;               * Actiuni                                              *
;               *       Ca la starea A8                                *
;               ********************************************************
                ajmp    StateA8                 ;Emisie ca slave
SendStart:      setb    sta                     ;Emisie START
                ajmp    ClrSiExit               ;Exit
                ds      2                       ;2 + 2 + 2 + 2
;
;               ********************************************************
;               * Stare C0:     (mod slave emisie)                     *
;               *       S-au transmis datele. S-a receptionat NACK     *
;               * Actiuni                                              *
;               *       Ca la starea A0                                *
;               ********************************************************
                ajmp    StateA0                 ;Ca la starea A0
;
SendStop:       setb    sto                     ;Emisie STOPStop
                clr     IICCntrl.MSTNOTREADY    ;Stergere bit \Ready
                ajmp    ClrSiExit               ;Exit
                                                ;2 + 2 + 2 + 2
;
;               ********************************************************
;               * Stare C8    (mod slave emisie)                       *
;               *       S-au transmis ultimele date (AA=0). Rec. ACK   *
;               * Actiuni                                              *
;               *       Ca la starea A0                                *
;               ********************************************************
                ajmp    StateA0                 ;Ca la starea A0
;
;               ******************************************
;               * Proceduri SIO1                         *
;               ******************************************
SendData:       cjne    r2,#0,ProceedSend          ;Emisie date daca exista
                jb      IICCntrl.INBLOCK2,SendStop ;Emisie STOP daca este bloc2
                acall   SetupBlock2                ;Altfel, set bloc2
                jnb     IICCntrl.SLREAD,SendData   ;Daca \mod citire, emisie bloc2
                ajmp    SendStart                  ;Altfel, emisie START repetat
;
ProceedSend:    dec     r2                      ;Decrementare contor date
                mov     a,@r1                   ;Date
                inc     r1                      ;Actualiyare pointer date
;
SendAccExit:    mov     s1dat,a                 ;emisie date
ClrSiExit:      clr     si                      ;sterge SI
SIO1Exit:       pop     acc                     ;reface Acc
                pop     psw                     ;reface PSW
                reti                            ;Exit
;               ********************************************************
;               * Init_Slave                                           *
;               ********************************************************
Init_Slave:     ret
;
;               ********************************************************
;               * Receive_Slave                                        *
;               *                                                      *
;               * Actiuni                                              *
;               *       If IICCntrl ( Byte 1 Expected ) Then           *
;               *         IICCntrl ( Byte 1 Expected ) := 0            *
;               *         SubAddress := s1dat                          *
;               *       Else                                           *
;               *         @SubAddress := s1dat                         *
;               *         SubAddress                                   *
;               ********************************************************
Receive_Slave:  mov     a,s1dat                         ;Data receptionata
                mov     r0,#r17                         ;Indicare subadresa
                jbc     IICCntrl.BYTE1EXPECTED,SaveData ;Daca octet1, salveaza in subadresa
```

```
                mov     r0,r17                      ;Altfel, @subadresa
                inc     r7                          ;Post-incrementare subadresa
SaveData:       mov     @r0,a                       ;Salvare date
                ret
;
;               *****************************************************
;               * Send_Slave                                        *
;               *                                                   *
;               * Actiuni                                           *
;               *       s1dat := @SubAddress                        *
;               *       SubAddress ++                               *
;               *****************************************************
Send_Slave:     mov     r0,r17
                mov     s1dat,@r0
                inc     r7
                ret
;
                end     ;
```

2.9. Timere de viteză mare

În multe aplicaţii este necesară implementarea de contoare soft. În continuare se prezintă doua rutine pentru implementarea a 8 contori soft. Ambele funcţii primesc un octet şi, în funcţie de starea biţilor din acest octet sau în funcţie de starea anterioara a acestora, se incrementează sau nu contorul aferent. Prima funcţie 'count' implementează 8 contoare pe 32 de biţi care pot fi setate independent ca să numere crescător sau descrescător, pe frontul crescător sau descrescător şi cu marcarea depăşirii capacităţii de numărare şi a modificării valorii. A doua funcţie implementează 8 contoare care pot număra numai crescător sau descrescător cu marcarea modificării valorii contorului.

```
/***********************************************************************\
**  Titlu:        COUNT.H                                             **
**  Descriere:    Implementează contoare soft.                        **
**                                                                    **
**  Versiune:     2.0                                                 **
**  Început la:   iunie, 1997                                         **
**  Autor:        Ştefan Suceveanu, Pratco s.r.l. Bucureşti, România  **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                  **
**                                                                    **
**  Headerul funcţiilor referitoare la contoare soft.                 **
**                                                                    **
**  Ultima modificare la data: 10 februarie 1998                      **
**  Istoric:                                                          **
\***********************************************************************/
#ifndef _COUNT_H_
#define _COUNT_H_   1

#include <typedef.h>       /* definitiile unor tipuri de date */

/* Definire stari contor */
#define C_ISNEW      0x01   /* valoare noua */
#define C_STATE      0x02   /* semnalului de intrare [C_ISUP/C_ISDOWN] */
#define C_ISUP       0x02   /* semnalul de intrare este 'sus' */
#define C_ISDOWN     0x00   /* semnalul de intrare este 'jos' */
#define C_OVERFLOW   0x04   /* depăşire contor */
#define C_COUNTDOWN  0x08   /* numără pe frontul descrescător */
#define C_RISINGEDGE 0x10   /* 1 = numără pe frontul crescător */

/* Definirea structurii unui contor */
typedef struct
{
       blword count;        /* valoarea contorului unsigned long int */
       byte   flags;        /* starea contorului */
} TCOUNTER;

#define N_COUNTERS   8      /* numarul de contoare - multiplu de 8 */

/* definire alocare memorie pentru contoare */
extern xdata TCOUNTER counters[N_COUNTERS];

/* definirea funcţiilor implementate */
void reset_count(byte n);          /* resetarea unui contor */
```

```
void reset_counts(void);          /* resetarea tuturor contoarelor */
void count(byte val);             /* implementeaza 8 contoare */
void quick_count(byte val);       /* implementare 8 contoare rapizi */

#endif
/***************************************************************************\
**  Titlu:        COUNT.C                                                  **
**  Descriere:    Implementează contoare soft.                             **
**                                                                         **
**  Versiune:     2.0                                                      **
**  Început la:   iunie, 1997                                              **
**  Autor:        Ştefan Suceveanu, Pratco s.r.l. Bucureşti, România       **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                       **
**  Copyright:    PRATCO s.r.l. Bucureşti, România,                        **
**                C.P. 61-137, RO 75500,                                   **
**                Tel./Fax: (+40)-1-345.17.25                              **
**                e-mail: office@pratco.ro                                 **
**                www:    www.pratco.ro                                    **
**                                                                         **
**  Acest modul conţine următoarele funcţii:                               **
**      void reset_count(byte n) = resetează contorul n                    **
**      void reset_counts(void)  = resetează toate contoarele              **
**      void quick_count(byte newval) = implementează 8 contoare           **
**                                                                         **
**  Ultima modificare la data: 10 februarie 1998                           **
**  Istoric:                                                               **
**                                                                         **
**  Copyright (C) 1998 PRATCO s.r.l. All rights reserved.                  **
\***************************************************************************/

#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
#include <typedef.h>
#include <stdio.h>
#include <reg552.h>
#include "count.h"

#define N_COUNTERS 8 /* nu modificaţi această valoare */

xdata TCOUNTER counters[N_COUNTERS];

/************************************************************************
        Resetează contorul n
*************************************************************************/
void reset_count(byte n)
{
   counters[n].count.Lw = counters[n].flags = 0;
}
/************************************************************************
        Resetează toate contoarele [N_COUNTERS]
*************************************************************************/
void reset_counts(void)
{
   byte i;

   for(i = 0; i < N_COUNTERS; i++)
      reset_count(i);
}
/************************************************************************
Implementează 8 contoare soft.
*************************************************************************/
void count(byte val)
{
  static data byte oldval;
  data byte i,sc,sd;
  data byte *pflags;
  data blword *pcx;

  sc = (oldval ^ newval) & newval;      // identificam fronturile crescătoare */
  sd = (oldval ^ newval) & oldval;      // identificam fronturile descrescătoare */

  for(i = 0; i < 8; i++)
  {
     pflags = &(counters[i].flags);
     pcx    = &(counters[i].count);

     if(*pflags & C_RISINGEDGE)         // numărăm pe frontul crescător
     {
        sd = sd << 1;                   // în CY mark front descrescător
        if(CY)
          *pflags ^= IS_UP;             // marcam noua stare

        sc = sc << 1;                   // în CY mark front crescător
        if(CY)
```

```
        {
          if(*pflags & C_COUNTDOWN)
            (*pcx)--;                  // decrementăm
          else
            (*pcx)++;                  // incrementăm

          *pflags |= C_ISNEW | (*pcx ? 0x00 : C_OVERFLOW);
          *pflags |= IS_UP;            // marcăm noua stare
        }
      }
      else                             // numărăm pe frontul descrescător
      {
        sc = sc << 1;                  // în CY mark front crescător
        if(CY)
          *pflags |= IS_UP;            // marcam noua stare

        sd = sd << 1;                  // în CY mark front descrescător
        if(CY)
        {
          if(*pflags & C_COUNTDOWN)
            (*pcx)--;                  // decrementam
          else
            (*pcx)++;                  // incrementam

          *pflags |= C_ISNEW | (*pcx ? 0x00 : C_OVERFLOW);
          *pflags ^= IS_UP;            // marcam noua stare
        }
      }
  }
}
/*************************************************************************
  Implementează 8 contoare soft. Valoarea newval este comparată bit cu
  bit cu valoarea anterioară şi dacă a apărut un front crescător se
  incrementează contorul corespunzător.
*************************************************************************/
void quick_count(byte newval)
{
  static data byte oldval;
  data byte s,i;

  // numara pe front crescator
  /* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  *** EXEMPLU ***
      front           ------  ---|          |---
              ------             v____  ____^
      oldval = 0000     0010     0010     0000
      newval = 0000     0010     0000     0010
              ------   ------   ------
      xor    = 0000     0000     0010     0010  -> '1' = un front
      newval = 0000     0010     0000     0010
              ------   ------   ------
      and    = 0000     0000     0000     0010  -> '1' = frontul crescător
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
  s = (oldval ^ newval) & newval;

  // numără pe front descrescător
  /* ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  *** EXEMPLU ***
      front           ------  ---|          |---
              ------             v____  ____^
      oldval = 0000     0010     0010     0000
      newval = 0000     0010     0000     0010
              ------   ------   ------
      xor    = 0000     0000     0010     0010  -> '1' = un front
      newval = 0000     0010     0010     0000
              ------   ------   ------
      and    = 0000     0000     0010     0000  -> '1' = frontul descrescător
  ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ */
  // s = (oldval ^ newval) & oldval;

  oldval = newval;

  if(s)
    for(i = 7; i; i++)
    {
       s = s << 1;                  // rotim la stiga prin CY
       if(CY)                       // daca CY=1 => front
       {
  counters[i].count.Lw++;  // incrementam
  // modificăm flagurile aferente contorului.
  counters[i].flags |= C_ISNEW;
/* se poate renunţa la linia de deasupra daca se doreşte viteza si mai mare */
       }
```

```
    }
}
```

O altă aplicaţie interesantă constă în utilizarea modulelor PWM de la circuitul 80C167. Acestea permit realizarea unui timer ultrarapid de 25 ns.

Principiul de lucru constă în interceptarea valorii registrelor de numărare PTx la fiecare eveniment extern. Deoarece numărătoarele PTx se pot seta să numere chiar cu frecvenţa procesorului (40 MHz), diferenţa între două valori indică diferenţa de timp în incremente de 25 ns.

Atenţie! Chiar dacă întreruperile externe rapide sunt eşantionate la 25 ns, arbitrarea şi prelucrarea lor este făcută tot 100 ns, de aici rezultând o eroare de 0...4 incremente de 25 ns.

2.10. Sinteză de frecvenţă

Una din cele mai des întâlnite aplicaţii ale microcontrolerelor este întâlnită în domeniul receptoarelor radio sau TV. Chiar larga dezvoltare a domeniului a impus dezvoltarea unor microcontrolere specializate care includ, pe lângă partea de logică şi control, şi elementele necesare prelucrării analogice a semnalului (mixere, AFI, oscilatorul local, divizoare analogice de frecvenţă etc.). Rolul microcontrolerului este de a controla numeric frecvenţa oscilatorului local dintr-un receptor heterodină, principalele avantaje fiind date de stabilitatea deosebită a acestuia, controlul numeric al frecvenţei, gabaritul redus al montajului etc.

Un microcontroler de uz general poate fi utilizat pentru controlul buclei unui oscilator PLL numai împreună cu un divizor analogic de frecvenţă, parametrii semnalului de radiofrecvenţă (amplitudine, putere, frecvenţă) făcând inadecvată utilizarea unor circuite digitale de divizare.

Principial, schema bloc a unui oscilator local cu sinteză de frecvenţă este prezentată în figura 3.13.

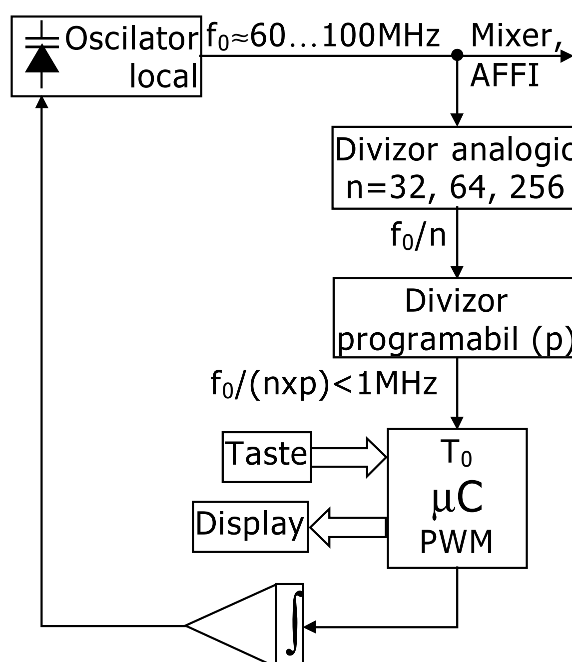


Figura 2.13. Schema unui bloc de sinteză de frecvenţă

Alegând în mod convenabil factorii de divizare ai divizorului analogic (de exemplu, pentru circuitul Sony MB506 aceştia pot fi 32, 64 sau 256, selectabili prin intermediul unor pini) şi ai divizorului digital, se poate stabili un ecart între două frecvenţe consecutive chiar şi de 0.1 Hz!

O altă posibilitate de realizare a unui tuner cu sinteză de frecvenţă constă în utilizarea unui circuit specializat, de exemplu TSA5055, circuit care se interfaţează cu microcontrolerul prin intermediul unei magistrale I^2C.

Aceste circuite sunt utilizate în special pentru tunere TV, controlul frecvenţei fiind asigurat de procesorul receptorului TV. Schema electrică de principiu poate fi găsită în catalogul corespunzător al firmei Philips.

Un program pentru utilizarea circuitului, precum şi al tastaturii şi display-ului pentru control, este prezentat în cele ce urmează.

```
/**********************************************************************\
**  Descriere:    Rutină comandă IIC TSA 5055                        **
**                                                                    **
**  Versiune:     1.0                                                 **
**  Inceut la:    Iunie 94                                            **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania  **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                  **
**                                                                    **
**  Acest modul contine urmatoarele functii:                          **
**                putTSA                                              **
**                getiic                                              **
**                                                                    **
**  Ultima modificare la data: 23 nov 1998                            **
**  Istoric:                                                          **
**                                                                    **
**  Observatii:                                                       **
**       foloseste rutine LCD                                         **
\**********************************************************************/
#include <reg552.h> /* SFR 80C552             */
#include <stdio.h>  /* standard I/O .h-file   */
#include <ctype.h>  /* standard I/O .h-file   */
#include <stdlib.h> /* standard functions     */
#include <string.h> /* string functions       */
#include <porturi.h>/* porturi I/O .h-file    */
#include <lcd.h>    /* subrutine LCD .h-file  */

/*scrie pe I2C,în componenta sel, la adresa adr, nr octeţi, variabila s*/
void putTSA(unsigned char sel,char nr,char *s);
/*citeşte pe I2C,din sel, de la adresa adr, nr octeţi, în variabila s*/
void getiic(char sel,char adr,char nr,char *s);

void main(void)
{
bit afis;
unsigned char i,tasta;
int divizor;
float frec,afisare;

char asir[10];

SCL = SDA = 1;             /* iniţializare SIO1 */
S1CON = 0xc1;

initLCD();
putLCD(0,0,"FRECVENTA RECEPTIE: ");
putLCD(2,0,"SELECTIE FRECVENTA: ");
afis = 1;
divizor = 3345;
afisare = ((float)divizor - 622) * 62.5;
asir[0] = 0x8e;
asir[1] = 0x50;
putTSA(1,2,&asir[0]);
while(1)
  {
   if(afis)
       {
       divizor = (divizor & 0x7fff);
       putTSA(1,2,&divizor);
       sprintf(asir,"%g",afisare);
       putLCD(1,0,"        ");
```

```
        putLCD(1,0,asir);
        cmdLCD(0xd4);
        cmdLCD(0x0d);
        afis = 0;
        }
        else
        {
        if(isdigit(tasta=kbds()))         /* setare factor */
               {
               putLCD(3,0,"       "); asir[0] = tasta;
               chrLCD(3,0,tasta); i = 1;
        while(i<8)
               {
               if((isdigit(tasta=kbds()))&&(i<7))
                      { asir[i] = tasta; chrLCD(3,i,tasta); i++;}
               if((tasta=='A')&&(i<7))
                  {
                      asir[i] = '.';
                      chrLCD(3,i,0x2e);
                      i++;
                      }
               if(tasta=='#')
                      {
                      asir[i] = 0; frec = atof(asir);
                      if((frec>44.9)&&(frec<860.1))
                             {
                             if(frec<170)  asir[1]=0x60;
                                    else if(frec<450)    asir[1]=0x50;
                                           else          asir[1]=0x30;
                             asir[0]=0x8e;
                             putTSA(1,2,&asir[0]);
                             divizor= (int)((frec*1000+38900.0)/62.5+0.5);
                             afisare=((float)((int)((frec*1000)/62.5+0.5))*62.5)/1000;
                             afis = 1;
                             break;
                             }
                      }
               if(tasta=='*')
                      {
                      tasta=kbds();
                      afis = 1;
                      putLCD(3,0,"       ");
                      break;
                      }
               }
        }
        if(tasta=='*')
               {
               putLCD(3,0,"       ");
               cmdLCD(0xd4);
               }
        }
    }
}

void putTSA(unsigned char sel,char nr,char *s)
{
bit gata = 0;
unsigned char i;
STA = 1; i = 0;

while(!gata)
   {
   while(!SI);
   switch(S1STA)
        {
        case 0:
               gata = 1;
               S1CON = 0xd5;
               break;
        case 0x08:
               S1DAT = 0xc0 | (sel << 1);
               S1CON = 0xc5;
               break;
        case 0x18:
               S1DAT = *(s+i);
               i++;
               S1CON = 0xc5;
               break;
        case 0x28:
               if(i<nr)
                      {
                      S1DAT = *(s+i);
```

```
                    i++;
                    S1CON = 0xc5;
                    }
             else
                    {
                    gata = 1;
                    S1CON = 0xd5;
                    }
             break;
      }
   }
}

void getiic(char sel,char adr,char nr,char *s)
{
bit gata = 0;
STA = 1;

while(!gata)
   {
    while(!SI);
    switch(S1STA)
      {
      case 0:
             gata = 1;
             S1CON = 0xd5;
             break;
      case 0x08:
             S1DAT = 0xa0 | (sel << 2);
             S1CON = 0xc5;
             break;
      case 0x10:
             S1DAT = 0xa1 | (sel << 2);
             S1CON = 0xc5;
             break;
      case 0x18:
             S1DAT = adr;
             S1CON = 0xc5;
             break;
      case 0x28:
             S1CON = 0xe5;
             break;
      case 0x40:
             if(nr==1) S1CON = 0xc1;
             else      S1CON = 0xc5;
             break;
      case 0x50:
             if(nr>2)
                    {
                    nr--;
                    *(s+nr) = S1DAT;
                    S1CON = 0xc5;
                    }
             else
                    {
                    *(s+1) = S1DAT;
                    S1CON = 0xc1;
                    }
             break;
      case 0x58:
             *s = S1DAT;
             S1CON = 0xd5;
             gata = 1;
             break;
      }
   }
}
```

2.11. Sisteme pentru controlul poziţiei

Determinarea poziţiei unui obiect poate fi uneori o problemă deosebit de complicată. Bineînţeles, fără a avea pretenţia construcţiei unui robot industrial, vor fi prezentate unele principii şi metode practice de determinare a poziţiei utilizabile pentru măsurarea unor poziţii unghiulare sau liniare.

Echipamentele industriale utilizate pentru măsurarea poziţiei au căpătat o mare dezvoltare în urma dezvoltării impetuoase a tehnicii de calcul.

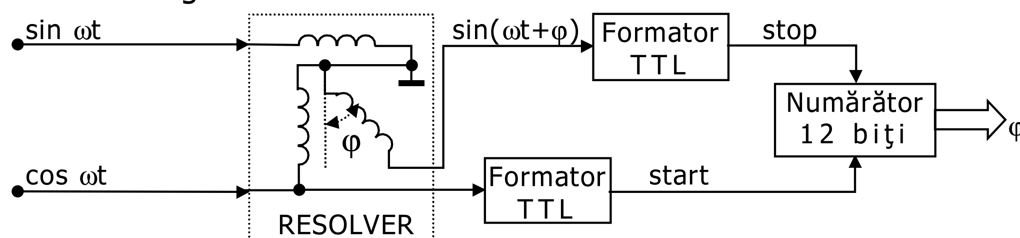
Măsurarea poziţiei, unghiulară sau liniară, este esenţială pentru imprimante, roboţi, automatizări industriale, maşini de prelucrat cu comandă numerică etc. Multiplele utilizări au făcut ca traductoarele de poziţie, dispozitive care convertesc poziţia într-o valoare numerică, să fie deosebit de numeroase.

Întrucât scopul lucrării nu este, de exemplu, prezentarea unui strung cu comandă numerică, vor fi descrise doar două tipuri de traductoare mai larg folosite: resolverul şi traductorul incremental.

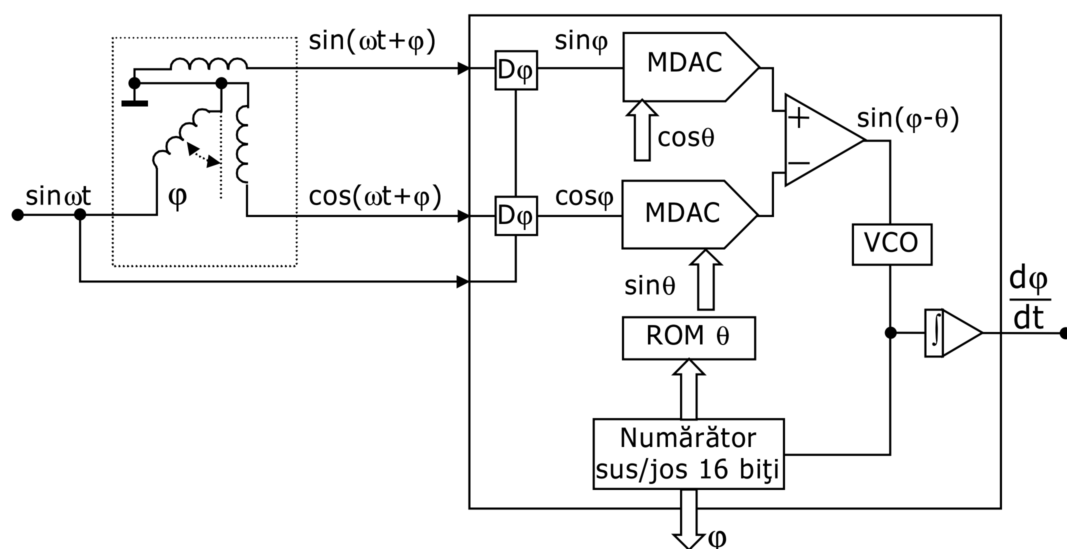
Resolverul este un traductor de poziţie unghiulară absolut care funcţionează pe principiul unui transformator rotativ. Cel mai simplu resolver este alcătuit dintr-un stator cu două înfăşurări defazate la 90°, alimentat cu două tensiuni alternative de excitaţie defazate la 90°, precum şi un rotor care culege o tensiune alternativă defazată faţă de tensiunile de excitaţie funcţie de poziţia unghiulară a rotorului faţă de stator.

Precizia de măsurare minimă asigurată de un resolver este de 12 biţi (2.63′), dar folosind scheme de măsurare în buclă închisă poate ajunge la 16 biţi (9.88″) sau chiar la 20 biţi (0.62″).

Cele două scheme de măsurare, în buclă deschisă şi închisă, sunt prezentate în Figura 2.14.



a) Măsurarea în buclă deschisă



b) Măsurarea în buclă închisă

Figura 2.14. Principii de măsurare a poziţiei resolverului

Funcţionarea schemei în buclă deschisă este relativ simplă, constând practic într-un cronometru care măsoară defazajul între un semnal de referinţă (în acest caz cos ωt) şi semnalul produs de resolver.

Montajul în buclă închisă este mai complex, funcţionarea sa bazându-se pe urmărirea unghiului φ, astfel încât acesta să fie egal cu unghiul θ. Cu excepţia avantajelor date de stabilitatea şi precizia ridicată, acest principiu de măsurare oferă şi un semnal direct proporţional cu viteza de rotaţie a resolverului, similar cu semnalul produs de un tahogenerator. Bucla care asigură performanţele acestui convertor este formată din două convertoare numeric/analogice cu multiplicare (MDAC) care produc tensiunile $\sin\varphi\cdot\cos\theta$, respectiv $\cos\varphi\cdot\sin\theta$, circuitul de scădere care produce tensiunea $\sin\varphi\cdot\cos\theta-\cos\varphi\cdot\sin\theta=\sin(\varphi-\theta)$, oscilatorul comandat în tensiune (VCO), numărătorul reversibil şi memoriile pentru $\sin\theta$ şi $\cos\theta$. Scopul buclei este de a asigura egalitatea între unghiul fizic φ şi unghiul virtual θ. Întârzierile produse în urmărirea de către unghiul θ a unghiului φ, direct proporţionale cu viteza de rotaţie a resolverului, sunt reliefate de frecvenţa VCO; această frecvenţă este transformată de integrator într-un semnal continuu care oferă informaţii despre viteza de rotaţie a resolverului.

Dacă prima schemă este destul de uşor de realizat (trebuie proiectate cu atenţie formatoarele TTL care nu trebuie să producă defazări mai mari de 50 ns), a doua schemă bloc este mult mai dificil de transpus în practică. Totuşi, există circuite monolitice sau hibride care funcţionează pe acelaşi principiu, de exemplu familia AD 2S8x produsă de firma Analog Devices.

În ceea ce priveşte traductoarele incrementale, cu toate că pot avea o precizie extrem de mare de până la 32 de biţi (adică 0.0003”), fiind de tip relativ necesită existenţa unei referinţe faţă de care se măsoară această poziţie. De fapt, această referinţă poate consta în cazul traductoarelor unghiulare într-un simplu contact acţionat de o camă.

Traductoarele incrementale sunt dispozitive profesionale, la fel de greu de procurat de amatori ca şi resolverele. Totuşi, există o soluţie extrem de economică aflată la îndemână oricărui amator: mouse-ul. În această situaţie singura problemă mai dificilă rămânând conectarea mecanică între mecanismul de controlat şi mecanismul mouse-ului. Bineînţeles, semnalele electrice de la mouse se conectează la microcontroler la o interfaţă serială, protocolul Microsoft serial mouse fiind prezentat în tabelul 3.25.

Tabelul 2.27

Octet	Bit						
1	1	LB	RB	Y7	Y6	X7	X6
2	0	X5	X4	X3	X2	X1	X0
3	0	Y5	Y4	Y3	Y2	Y1	Y0
4	0	MB	0	0	0	0	0

Interfaţa serială trebuie setată la următorii parametri: 1200 biţi/secundă, 7 biţi de date, fără paritate, un bit de stop. Semnificaţia biţilor din tabel este:

- primul bit este folosit pentru sincronizare;
- LB = buton stânga – BYTE1 and 0x20;
- RB = buton dreapta – BYTE1 and 0x10;
- MB = buton mijloc – BYTE4 and 0x20;
- deplasare X = (BYTE1 and 0x03) << 6 + BYTE2;

- deplasare Y = (BYTE1 and 0x0c) << 4 + BYTE3.

Până acum, traductoarele de poziţie au fost folosite numai pentru măsurarea unor unghiuri. Deplasările liniare sunt măsurabile prin cuplarea la axul traductorului unghiular a unui disc. Astfel, deplasarea liniară a discului este transformată în deplasare unghiulară a traductorului.

3. Accesorii pentru sistemele cu microcontrolere

În afară de microcontroler şi circuitele sale de suport necesare (memorii externe, display, tastatură, diferite periferice I^2C etc.), un produs mai poate avea nevoie şi de alte dispozitive: surse de alimentare, interfeţe seriale particularizate, decodificatoare, numărătoare suplimentare etc.

3.1. Surse de alimentare

În general, sursele de alimentare utilizate pentru montajele electronice cu microcontrolere necesită o tensiune continuă de alimentare de 5V / 1W şi, eventual, funcţie de prezenţa unor circuite analogice, a altor tensiuni pentru alimentarea acestora (de regulă –5V, ±12V, ±15V etc.).

Sursele de alimentare se pot clasifica în două mari categorii:

- surse liniare, la care tensiunea de ieşire este obţinută din tensiunea de intrare pe baza unui circuit liniar (filtru, stabilizator cu reacţie negativă etc.);
- surse în comutaţie, la care tensiunea de ieşire este obţinută prin prelucrarea unei tensiuni alternative cu o frecvenţă de până la câteva sute de kiloherţi, tensiune alternativă obţinută din tensiunea de intrare.

Cu toate că sursele în comutaţie sunt net superioare surselor liniare din toate punctele de vedere, acestea din urmă se mai folosesc pentru montaje nepretenţioase, mai ales datorită simplităţii lor.

3.1.1 Surse liniare

Schema bloc generală a unei surse liniare este prezentată în figura 4.1.

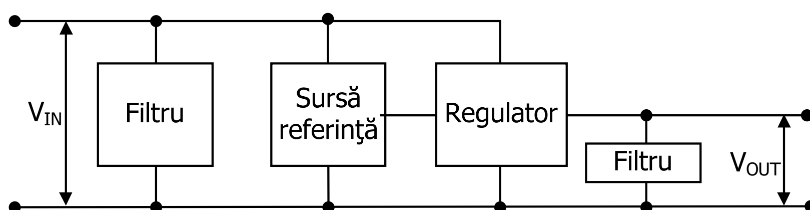


Figura 3.1. Sursă liniară de tensiune

Caracteristicile generale ale surselor liniare (stabilizarea în raport cu variaţia tensiunii de intrare sau a curentului de sarcină, factorul de rejecţie a ondulaţiei tensiunii de intrare, stabilitatea funcţie de temperatură etc.) depind de elementele schemei şi, pentru obţinerea unor parametri calitativi, este necesară o proiectare laborioasă. Un alt dezavantaj al surselor liniare este randamentul scăzut, precum şi faptul că nu se pot obţine decât tensiuni de aceeaşi polaritate şi mai mici cu tensiunea de intrare.

Din punct de vedere practic, sursele liniare sunt fezabile pentru montaje nepretenţioase, utilizarea surselor liniare monolitice din seria LM78xx, LM3xx etc. asigurând o simplitate deosebită.

Câteva montaje pentru obţinerea tensiunii de alimentare de 5V sunt prezentate în figura 4.2.

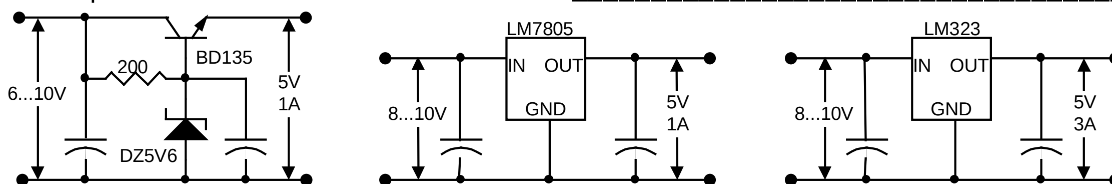


Figura 3.2. Surse liniare de 5V.

3.1.2 Surse în comutaţie

De regulă, sursele de alimentare în comutaţie produc tensiunea de ieşire dintr-o tensiune alternativă de câteva sute de kiloherţi generate de un oscilator din tensiunea de alimentare.

Principalele avantaje ale surselor în comutaţie sunt randamentul ridicat (ajunge până la 97%), dar şi obţinerea de tensiuni cu orice polaritate şi valoare faţă de tensiunea de intrare. De asemenea multe din circuitele utilizate mai au şi alte facilităţi, cum ar fi: detectarea scăderii tensiunii de intrare şi generarea unui semnal $\overline{NMI}$, pornirea/oprirea sursei prin intermediul unui semnal de comandă TTL etc.

În figura 4.3 sunt prezentate două tipuri de surse de 5V: una din ele este destinată folosirii în aparatele portabile, tensiunea de 5V de mică putere (240mA) fiind produsă pentru tensiuni de intrare între 1V şi 6.2V; cea de doua este o sursă de putere mare (5A) utilizabilă pe autovehicule (tensiunea de intrare în gama 10...60V).

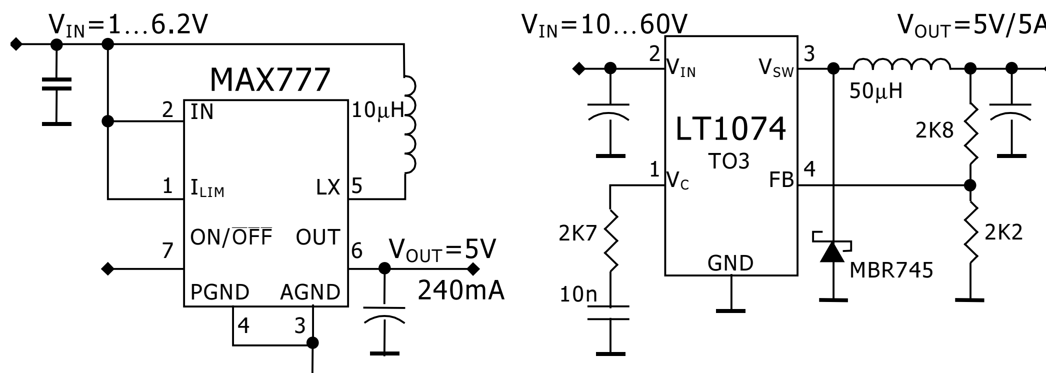


Figura 3.3. Surse de alimentare în comutaţie

O altă problemă care poate fi dificil de rezolvat în situaţia dispozitivelor staţionare (alimentate la reţeaua 220V/50Hz) care au impus un gabarit redus îl constituie transformatorul coborâtor, de la 220V la 8...10V. Este posibilă utilizarea unor divizoare rezistive sau capacitive care, bineînţeles, nu mai oferă siguranţa oferită de separarea de reţea prin intermediul transformatorului.

O astfel de sursă, construită în jurul circuitului MAX610, este prezentată în figura 4.4.

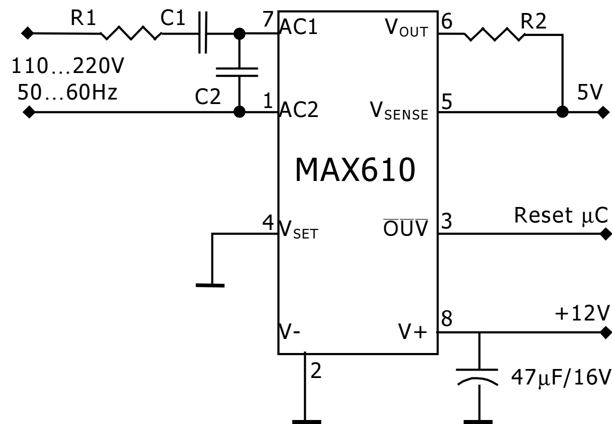


Figura 3.4. Sursă liniară la 220V/50Hz

Componentele pasive C1, C2, R1 şi R2 se determină conform cu relaţiile următoare:

- $C1 = \frac{I_{OUT}}{4\sqrt{2}\cdot(V_{IN} - V_{OUT})\cdot f_{IN}}$;

  Pentru descărcarea C1 se recomandă montarea în paralel a unei rezistenţe de 1M.
- C2=47µF;
- R1=100Ω (pentru protecţia C2);

  $P_d(R1)[mW]=2.7\cdot C1^2[\mu F]\cdot R1[\Omega]$;
- $R2 = \frac{0.6}{I_{LIMIT}}$.

3.2. Interfeţe seriale

Interfaţa serială este o metodă, aproape universală, utilizată pentru schimbul de date şi informaţii între microcalculatoare. Chiar dacă rata de transmisie a datelor nu este impresionantă, simplitatea componentelor hardware şi software necesare, au făcut totuşi din interfaţa serială procedura de bază pentru interconectare. Astăzi, antica RS-232 a fost ameliorată în sensul vitezei de transmisie, detectării şi corectării erorilor, creşterea distanţei de transmisie, majorarea numărului de circuite care sunt conectate la aceeaşi linie de transmisie etc.

3.2.1 Detectarea automată a vitezei de transmisie seriale

Pentru detectarea automată a vitezei de comunicaţie seriala se foloseşte primul caracter recepţionat. Algoritmul prezentat în AN447 (Philips) măsoară timpul necesar transmiterii unui caracter în format 8biţi, fără paritate, 1 bit de stop. Deoarece caracterele ASCII normale au bitul 7 zero şi deoarece transmisia seriala începe cu bitul 0 şi se termină cu bitul 7, se poate determina începutul bitului de stop.

Algoritmul de mai jos aşteaptă bitul de start (un front căzător) pe pinul de recepţie serială şi porneşte timerul `T0`. La fiecare front crescător se citeşte valoarea timerului `T0` şi se memorează. Când apare întreruperea de depăşire

(*overflow*) a timerului `T0`, ultima valoare capturată va indica durata transmisiei unui caracter, de la bitul de start până la bitul de stop.

În tabelul `CmpTable` se găsesc valorile maxime ale duratei unui caracter pentru fiecare viteză de transmitere. Trebuie avut în vedere ca să nu se transmită un al doilea caracter imediat după primul.

Dacă se foloseşte şi bitul de paritate, pot apare erori de determinare a vitezei de comunicaţie dacă se transmite oricare din caracterele de la 'p' la 'z', acoladele '{' sau '}', bara verticală '|', tilda '~' sau caracterul 'delete', în funcţie de paritatea folosită (pară sau impară). Caracterele uzuale folosite pentru a atrage atenţia sistemului (spaţiu, enter, escape) nu prezintă aceste neajunsuri.

Trebuie avut în vedere, de asemenea, că primul caracter recepţionat este pierdut, neputând fi identificat corect.

De asemenea dacă după determinarea vitezei de transmisie se constată erori de încadrare (*framing errors*) atunci trebuie repetat algoritmul de determinare a vitezei.

Pentru a calcula valorile din tabelul `CmpTable` se foloseşte următoarea formulă:

$$\text{Valoare din tabel} = \frac{\text{Osc[MHz]}}{\text{Viteza de transmitere}} \cdot \frac{5}{12}$$

Valorile din tabel sunt pe 16 biţi şi deci rezultatele de mai sus trebuie împărţite în două după octetul superior şi inferior.

Formula de mai sus a fost dedusă folosind următoarele ecuaţii:

$$\text{valoarea maxima din tabel} = \frac{\text{timpul minim de recunoastere}}{\text{durata unui ciclu masina}}$$

$$\text{timpul minim de recunoastere} = \frac{\text{nr. maxim de biti de recunoscut}}{\text{numarul de biti vizibili}} \cdot \text{durata unui octet}$$

numărul de biţi vizibili este 9 şi numărul de biţi de recunoscut este 5 pentru 8-N-1.

$$\text{durata unui octet} = \frac{1}{\text{Viteza de transmitere}} \cdot \text{numarul de biti vizibili}$$

$$\text{durata unui ciclu masina} = \frac{\text{frecventa oscilatorului}}{12}$$

```
/**********************************************************************\
**  Descriere:    Detectarea automată a vitezei de transmisie         **
**                                                                    **
**  Versiune:     1.0                                                 **
**  Inceut la:    Iunie 94                                            **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania  **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                  **
**                                                                    **
**  Acest modul contine urmatoarele functii:                          **
**                                                                    **
**  Ultima modificare la data: 23 nov 1998                            **
**  Istoric:                                                          **
**                                                                    **
**  Observatii:                                                       **
**                                                                    **
\**********************************************************************/
$Mod552
```

```
RX          BIT       P3.0         ;Pin recepţie.
CharH       DATA      30h          ;Octet superior rezultat timer.
CharL       DATA      31h          ;Octet superior rezultat timer.
BRate:      DATA      32h          ;Valoare finală viteză de transmisie.
Display     EQU       P4           ;Port afişare pentru depanare.

;           Vectori întreruperi şi reset
ORG 8000h
START:      ACALL     AutoBaud     ;determinare viteză transmisie
MOV         Display,Brate          ;afişare viteză
SJMP        START

;           Subrutine
;Subrutină determinare automată viteză de transmitere
;Detectează viteza de transmitere după primul caracter recepţionat prin măsurarea
;lungimii caracterului. Unele caractere pot să producă erori, în special cele care
;conţin la ;sfârşit 0X7h sau 0XFh.
AutoBaud:
            MOV       TMOD,#01h    ;Initializare timer T0
            MOV       TH0,#0
            MOV       TL0,#0
            MOV       TCON,#0

            MOV       CharH,#0     ;Initializare resultat timer.
            MOV       CharL,#0

AB0:        JB        RX,AB0       ;Aşteptare bit de start.
            SETB      TR0          ;Start timer.

AB1:        JB        TF0,AB3      ;Verificare overflow timer.
            JNB       RX,AB1       ;Verificare front crescător pe RxD.
            MOV       CharH,TH0    ;Capturare valoare timer.
            MOV       CharL,TL0    ;

AB2:        JB        TF0,AB3      ;Verificare overflow timer.
            JB        RX,AB2       ;Verificare front descrescător pe RxD.
            SJMP      AB1          ;Repetare măsurare.

AB3:        CLR       TR0          ;Depăşire timp maxim de măsurare; verificare resultat
            CLR       TF0          ;Oprire timer şi ştergere indicatori.

            MOV       BRate,#19    ;Setare tabel pointeri.

CmpLoop:    MOV       A,BRate
            MOV       DPTR,#CmpTable
            MOVC      A,@A+DPTR    ;Tabel comparare.
            DEC       BRate
            CJNE      A,CharH,Cmp1;Verificare rezultat.
            SJMP      CmpLow       ;Verificare octet inferior.
Cmp1:       JC        CmpMatch     ;Valoare tabel < valoare timp.
            DJNZ      BRate,CmpLoop;Verificare sfârşit tabel.
            SJMP      CmpMatch

CmpLow:     MOV       A,BRate
            MOVC      A,@A+DPTR    ;Tabel comparare.
            CJNE      A,CharL,Cmp2;Verificare rezultat
            SETB      C            ;Valoare tabel = valoare timp
Cmp2:       JC        CmpMatch     ;C setat dacă A < octet inferior ;rezultat.
            DJNZ      BRate,CmpLoop;Verificare sfârşit tabel.
CmpMatch:
            MOV       A,BRate        ;Comparare completă
            CLR       C
            RRC       A
            MOV       BRate,A        ;Salvare valoare
            RET

;Tabel comparare pentru valori timer. Ordinea: LSB, MSB.
;Valorile sunt pentru 12 Mhz.
CmpTable:
            DB        40h,00h        ;0 – valoare prea mică.
            DB        80h,00h        ;1 - 38,400 baud.
            DB        00h,01h        ;2 - 19,200 baud.
            DB        00h,02h        ;3 -  9,600 baud.
            DB        00h,04h        ;4 -  4,800 baud.
            DB        00h,08h        ;5 -  2,400 baud.
            DB        00h,10h        ;6 -  1,200 baud.
            DB        00h,20h        ;7 -    600 baud.
            DB        00h,40h        ;8 -    300 baud.
            DB        00h,80h        ;9 – valoare prea mare.
```

3.2.2 Implementarea unei transmisii seriale cu pachete CRC16

În cazul transmiteri datelor în medii poluate electromagnetic pot apare erori. Aceste erori pot fi detectate folosind un algoritm simplu ca de exemplu suma fără transport a tuturor octeţilor dintr-un pachet, fie algoritmi mai complicaţi care asigură o probabilitate foarte mare de detectare a erorilor. Printre aceşti algoritmi se aflã şi Codul Ciclic Redundant (CRC). Algoritmul de determinarea a sumei de control propus determină CRC 16 folosind polinomul CRC-CCITT $x^{16} + x^{12} + x^{5} + x^{0}$ folosit în protocolul XMODEM, pentru un buffer 'buff' de lungime 'len'. Valoare pe 16 biţi obţinută trebuie adăugată la sfârşitul bufferului ce urmează a fi transmis astfel:

`Buff` *`High`*`(`**`CRC`**`)` *`Low`*`(`**`CRC`**`)`

La recepţie se calculează CRC16 pentru întreg bufferul recepţionat (inclusiv cu CRC-ul transmis → `len+2`) şi dacă valoarea obţinută este zero atunci transmisia s-a făcut fără erori. Dacă valoarea CRC-ului calculat este diferită de zero atunci au apărut erori la transmisie şi în funcţie de protocolul de comunicaţie folosit se poate transmite un NACK sau nu se transmite nimic.

Funcţia `unsigned int bufCRC16(char *buff, char len, unsigned int CRC)` are implementat calculul CRC-ului pentru cele două moduri de memorare a valorilor întregi în memorie (**litle endian →** procesoarele Intel x86, Pentium sau **big endian →** procesoarele Motorola, Philips). Această funcţie primeşte ca parametrii un pointer la şirul cu date care urmează să fie transmisă, `buff`, lungimea acestui şir, `len`, şi valoarea `CRC`-ului de la care se pleacă. Dacă se doreşte calculul CRC-ului numai pentru şirul indicat de buff atunci parametrul `CRC` se va iniţializa cu 0. Dacă se doreşte concatenarea a două şiruri valoarea CRC-ul obţinut după apelul acestei funcţii pentru primul şir de caractere se va pasa ca parametru la apelul acestei funcţii pentru al doilea şir de caractere.

Pentru creşterea vitezei, în memoria program se memorează un tabel cu codurile CRC16 pentru toate valorile unui octet, tabel care are lungimea de 512 octeţi.

```
/***************************************************************************\
**  Descriere:    Calculeaza CRC16 pe baza polinomului CRC-CCITT (XMODEM)    **
**                                                                           **
**  Versiune:     2.0                                                        **
**  Inceut la:    Iunie 1998                                                 **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania         **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                         **
**  Copyright:    PRATCO s.r.l. Bucuresti, Romania,                          **
**                C.P. 61-137, RO 75500, Bucuresti, Romania                  **
**                Tel./Fax: (+40)-1-345.09.75                                **
**                e-mail: office@pratco.ro                                   **
**                www:    www.pratco.ro                                      **
**                                                                           **
**  Acest modul contine urmatoarele functii:                                 **
**     unsigned int bufCRC16(char *buff, char len, unsigned int CRC)         **
**                                                                           **
**  Ultima modificare la data: 24 iunie 1998                                 **
**  Istoric:                                                                 **
**                                                                           **
**  Observatii: Pentru ca calculul CRC-ului să fie independent de procesorul **
**  folosit trebuie modificat in typedef.h funcţiile LOW si HIGH care extrag **
**  din memorie partea inferioara si respectiv superioara a unui intreg pe   **
**  16 biti.                                                                 **
**  Pentru verificarea CRC-ului la receptie trebuie ca CRC-ul unui mesaj la  **
```

```
**  care s-a adăugat la sfârşit HIGH(CRC) si LOW(CRC) trebuie sa fie zero.   **
**  Copyright (C) 1998 PRATCO s.r.l. All rights reserved                     **
\****************************************************************************/
#pragma DEBUG OBJECTEXTEND CODE SYMBOLS
#include <typedef.h>

/* TABEL CRC-CCITT x^16 + x^12 + x^5 + 1 */

code unsigned int crctab[256] = {
/*00*/  0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7, /*07*/
/*08*/  0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF, /*0F*/
/*10*/  0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6, /*17*/
/*18*/  0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE, /*1F*/
/*20*/  0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485, /*27*/
/*28*/  0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D, /*2F*/
/*30*/  0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4, /*37*/
/*38*/  0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC, /*3F*/
/*40*/  0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823, /*47*/
/*48*/  0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B, /*4F*/
/*50*/  0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12, /*57*/
/*58*/  0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A, /*5F*/
/*60*/  0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41, /*67*/
/*68*/  0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49, /*6F*/
/*70*/  0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70, /*77*/
/*78*/  0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78, /*7F*/
/*80*/  0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F, /*87*/
/*88*/  0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067, /*8F*/
/*90*/  0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E, /*97*/
/*98*/  0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256, /*9F*/
/*A0*/  0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D, /*A7*/
/*A8*/  0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405, /*AF*/
/*B0*/  0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C, /*B7*/
/*B8*/  0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634, /*BF*/
/*C0*/  0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB, /*C7*/
/*C8*/  0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3, /*CF*/
/*D0*/  0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A, /*D7*/
/*D8*/  0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92, /*DF*/
/*E0*/  0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9, /*E7*/
/*E8*/  0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1, /*EF*/
/*F0*/  0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8, /*F7*/
/*F8*/  0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0  /*FF*/
};

\**************************************************************************
  calculează CRC16 pentru un buffer de lungime len plecând de la un CRC
  existent pentru concatenarea CRC-ului a două buffere. La primul apel CRC
  trebuie sa fie 0.
**************************************************************************/
unsigned int bufCRC16(char *buff, char len, unsigned int CRC)
{
  data unsigned char i;

  for(i = 0; i < len; i++)
     // calcul CRC independent de procesor
     CRC = (((unsigned int)LOW(CRC)) << 8) ^ crctab[HIGH(CRC) ^ buff[i]];
     // calcul CRC pentru procesoare tip Intel (little endian)
//   CRC = (CRC << 8) ^ crctab[(CRC >> 8) ^ buff[i]];

  return CRC;
}
```

3.2.3 Sistem de transmisie cu curenţi purtători

O aplicaţie interesantă poate fi utilizarea reţelei electrice de 220V/50Hz ca mediu de transmisie a datelor. Informaţia, care poate consta chiar într-un pachet de date transmise serial, modulează în frecvenţă o purtătoare de circa 10 kHz.

Filtrul format din capacitatea C şi transformatorul reglabil pe ferită Tr asigură atât separarea de reţea cât şi adaptarea la linia de transmisie.

Întrucât mediul de transmisie este, prin definiţie, saturat cu paraziţi, este necesară adoptarea unor măsuri specifice pentru recepţionarea corectă a datelor seriale transmise: în primul rând reducerea vitezei de transmisie; în al doilea rând trebuie utilizate protocoale specifice pentru detectarea sau

autocorecţia erorilor. Schema electrică de principiu construită cu ajutorul a câte unui circuit CD4046 este prezentată în figura 4.5.

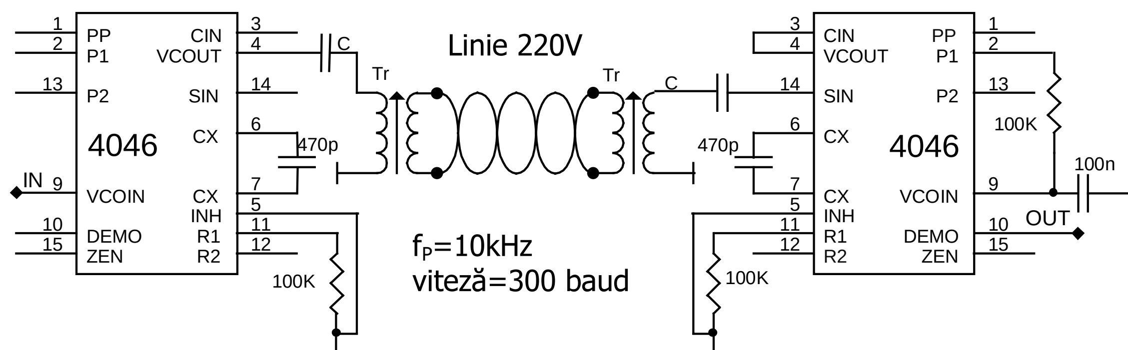


Figura 3.5. Interfaţă pentru transmisie cu curenţi purtători

3.2.4 Interfaţa CAN

Pentru distanţe mici, de câţiva metri, semnalele digitale care nu au o bandă prea mare se pot transmite direct, fără precauţii deosebite.

În cazul transmiterii informaţiilor la distanţe mari, este obligatorie folosirea unor circuite speciale, interfeţe, care realizează o amplificare a semnalului astfel încât atenuarea semnalului produsă de firul telefonic să nu reducă semnificativ raportul semnal zgomot. De regulă, amplificarea este realizată în tensiune sau în curent.

Aceste interfeţe sunt standardizate, principalele tipuri şi caracteristicile lor fiind prezentate în tabelul 4.1.

Tabelul 3.1

Tip interfaţă	Distanţă legătură	Număr emiţătoare pe un canal	Număr receptoare pe un canal	Semnal
RS 232	7 m	1	1	tensiune
RS 422	1200 m	1	32	tensiune
RS 485	1200 m	32	32	curent

Cu excepţia acestor tipuri de interfeţe, devenite deja clasice, în ultimii ani a apărut un protocol pentru interfeţe care operează în medii cu puternice perturbaţii electrice asigurând o protecţie ridicată împotriva erorilor, având o arhitectură deschisă, un mediu de transmisie cu proprietăţi definite de utilizator, într-un cuvânt interfaţa CAN (*Controller Area Network*). CAN este un sistem de magistrală serială performantă destinată controlului distribuit, destinată iniţial utilizării pentru autovehicule. Interfaţa a fost creată de firma Robert Bosch GmbH la sfârşitul anilor 1980.

Dezvoltarea CAN a fost impusă de implementarea unui număr din ce în ce mai mare de dispozitive electronice în autovehiculele moderne, cum ar fi: sisteme de control ale motorului, suspensii active, ABS, cutii de viteze automate, controlul luminilor, aer condiţionat, air-baguri, închidere centralizată etc. Toate acestea semnifică nu numai o siguranţă şi un confort

sporit pentru şofer, ci şi o reducere a consumului de carburant şi a emisiilor de noxe.

Îmbunătăţirea comportamentului autovehiculelor a făcut necesar ca diferitele sisteme de control, inclusiv senzorii lor, să schimbe informaţii. Schimbul de date era rezolvat prin interconectarea punct la punct a sistemelor. Între timp, necesităţile au crescut aşa de mult astfel încât era necesar un cablaj electric de mai mulţi kilometri şi o mulţime de conectoare. Toată această structură de trasee electrice prin autovehicul a creat probleme în ceea ce priveşte productivitatea, costul de producţie şi nu în ultimul rând fiabilitatea sistemului.

Soluţia acestei probleme a fost interconectarea sistemelor de control prin intermediul unei interfeţe seriale care, bineînţeles, avea caracteristici specifice datorită utilizării într-un autovehicul. Simplificarea cablajului electric impune, totuşi, adăugarea de dispozitive specifice la fiecare sistem din autovehicul, dispozitive care "cunosc" regulile şi protocolul de transmitere şi recepţionare a datelor pe interfaţa serială.

Interfaţa CAN este larg folosită pentru autovehicule şi automatizări industriale. Alte utilizări comune ale CAN sunt echipamente electronice pentru trenuri, tehnică medicală, automatizări casnice, sisteme de control securitate, mediu etc. pentru clădiri etc. În anul 1998 erau estimaţi circa 20 milioane de utilizatori CAN urmând ca după anul 2000 numărul acestora să se ridice la peste 140 milioane.

Cu excepţia interfeţei CAN dezvoltată de Bosch şi-au dezvoltat propria interfaţă şi alte companii auto: Volkswagen (A-BUS), Peugeot şi Renault (VAN – *Vehicle Area Network*), Chrysler, General Motors şi Ford (J1850) dar, de departe, dominaţia pe piaţa europeană aparţine CAN.

CAN este standardizată internaţional ISO (*International Standardization Organization*) şi SAE (*Society of Automotive Engineers*).

În concluzie, CAN este o magistrală multi-master, cu o structură liniară, deschisă, cu noduri egale. Protocolul nu limitează numărul de noduri. Numărul de noduri poate fi schimbat dinamic, fără a deranja funcţionarea celorlalte noduri.

a) Concepte de bază CAN

Conform cu modelul ISO-OSI, protocolul CAN are mai multe nivele:

- Nivel aplicaţie.
- Nivel obiect:
  - filtrare mesaje;
  - manipulare mesaje şi stări.
- Nivel transfer:
  - limitare erori;
  - detectare şi semnalare erori;
  - validare mesaje;
  - confirmare mesaje;

- arbitrare;
- încadrare mesaje;
- viteza de transfer şi sincronizare.

Nivelul de transfer constituie nucleul protocolului CAN. El prezintă mesajele recepţionate nivelului obiect şi acceptă de la acesta mesajele de transmis. Nivelul de transfer este responsabil cu sincronizarea biţilor, încadrarea mesajelor, confirmări, arbitrări, detectarea, semnalarea şi limitarea erorilor.

- Nivel fizic:
  - nivel semnal şi reprezentare biţi;
  - mediu de transmisie.

b) Caracteristici generale

Mesaje – informaţia este transmisă pe magistrală într-un format fix cu număr diferit (dar limitat) de biţi. Când magistrala este liberă, orice dispozitiv conectat la magistrală poate iniţia transmiterea unui mesaj.

Sincronizare – transferul datelor este corect dacă toleranţa oscilatorului de serializare a datelor este mai bună de 1.58%.

Traseul informaţiilor – în sistemele CAN, dispozitivele nu folosesc informaţia referitoare la configuraţia sistemului (de exemplu adresele dispozitivelor). Aceasta are câteva consecinţe importante:

- Flexibilitatea sistemului – reţelei CAN îi pot fi adăugate noi noduri fără a fi necesare modificări ale programelor sau echipamentelor nodurilor şi nivelului de aplicaţie.
- Rutarea informaţiilor – conţinutul unui mesaj este stabilit de un identificator (*Identifier*). Acesta nu arată destinaţia mesajului, dar descrie scopul datelor, orice nod din reţea fiind apt să decidă prin filtrarea mesajelor (*Message Filtering*) oricând informaţia îi este utilă sau nu.
- Recepţia multiplă – ca o consecinţă a filtrării mesajelor, orice nod din reţea poate recepţiona şi folosi simultan acelaşi mesaj.
- Consistenţa datelor – într-o reţea CAN este asigurată primirea unui mesaj fie de mai multe noduri, fie de nici unul. Consistenţa datelor este realizată de conceptul de recepţie multiplă şi de manipularea erorilor.
- Viteza de transmisie – poate fi diferită în sisteme CAN diferite. Totuşi, într-un sistem această viteză este fixă.
- Priorităţi – identificatorul defineşte o prioritate statică a mesajului pe durata accesului magistralei.
- Cererea de date – prin transmiterea unei cereri de date (*Remote Frame*), un nod poate cere date (*Data Frame*) altui nod. Atât *Data Frame* cât şi *Remote Frame* sunt definite de acelaşi identificator.
- Multimaster – dacă magistrala este liberă, orice nod poate iniţia transmiterea unui mesaj.
- Arbitrarea - dacă există simultan mai multe cereri de acces la magistrală, arbitrarea este câştigată de mesajul cel mai prioritar pe baza

identificatorului. Mecanismul de arbitrare asigură ca în nici un moment nici o informaţie să nu fie pierdută.

- Securitate – pentru a permite o siguranţă extremă a transferului de date, în fiecare dispozitiv CAN sunt luate măsuri deosebite pentru detectarea semnalarea şi autocontrolul erorilor. Detectarea erorilor constă în următoarele proceduri: monitorizarea magistralei (emiţătorul compară bitul transmis cu bitul detectat pe magistrală), coduri CRC, controlul cadrelor de mesaje şi biţii de adaus (pentru a garanta transmiterea datelor cu cod NRZ, dacă într-un mesaj sunt mai mult de 5 biţi identici, se va insera automat un bit complementar în fluxul de date).

  Controlul cu bit de adaus este asigurat pentru cadrele, câmpurile sau biţii: start cadru, arbitrare, control, date şi secvenţa CRC. Celelalte componente ale mesajului au o formă fixă şi nu suportă acest tip de control.

  Detectarea erorilor asigură o probabilitate de nedetectare a mesajelor corupte mai mică de (rata erorilor mesajului) $\cdot$ $4.7 \cdot 10^{-11}$;
- Semnalarea erorilor - este realizată de orice nod care le detectează. Mesajul eronat este eliminat şi va fi retransmis automat.
- Limitarea erorilor – este înfăptuită de nodurile CAN care pot face diferenţa între perturbaţiile aleatoare şi defectele permanente ale liniei sau circuitelor. Nodurile defecte sunt decuplate automat.
- Legături – interfaţa serială CAN constă într-o magistrală la care pot fi conectate un număr nelimitat de dispozitive. Practic, numărul total de noduri este limitat de întârzieri şi încărcarea liniei.
- Valorile magistralei – constau în două valori logice complementare denumite dominantă şi recesivă. Pe durata unor transmisii simultane a unui bit recesiv şi a unuia dominant, starea magistralei va fi dominantă.
- Confirmarea mesajelor – toate receptoarele verifică consistenţa mesajului confirmând un mesaj valid şi indicând un mesaj eronat.
- Modul inactiv/atenţionare – pentru reducerea consumului, un dispozitiv CAN poate intra în mod inactiv deconectând amplificatoarele de linie. Modul inactiv este întrerupt printr-o atenţionare: orice activitate pe magistrală sau alte condiţii interne ale circuitului.

c) Tipuri de cadre

Transferul mesajelor este efectuat şi controlat de cinci tipuri diferite de cadre:

Cadru de date (*Data Frame*) care transportă datele de la emiţător la receptor;

Cadru cerere de date (*Remote Frame*) este transmisă de un nod care solicită transmiterea unui cadru de date cu acelaşi identificator;

Cadru eroare (*Error Frame*) este transmis de orice dispozitiv care a detectat o eroare;

Cadru supraîncărcare (*Overload Frame*) este folosit pentru asigurarea unei întârzieri suplimentare între cadrele de date sau cerere de date.

Cadrele de date sau cerere de date sunt separate de celelalte cadre printr-un spaţiu inter-cadre (*Interframe Space*).

Cele cinci tipuri de cadre standard sunt prezentate în figura 4.6.

Semnificaţia câmpurilor, biţilor şi a altor termeni din figură este următoarea:

- Start cadru marchează începutul unui cadru de date sau cerere de date. Este folosit de toate nodurile pentru sincronizare.
- Câmpul de arbitrare constă în identificator (11 biţi, din care biţii 10…4 nu trebuie să fie toţi regresivi) şi bitul RTR (este dominant pentru cadrele de date, respectiv recesiv pentru cadrele cerere de date). Pentru a fi păstrată compatibilitatea cu standardul CAN standard, standardul CAN extins 29 biţi are identificatorul împărţit în două: identificator de bază şi identificator extins între care se inserează biţii SRR (un bit recesiv care este pe poziţia RTR; este folosit pentru a prevala cadrul standard în faţa unui cadru extins în cazul unei coliziuni) şi IDE (folosit pentru deosebirea dintre un cadru extins şi un cadru standard, situaţie în care este suprapus cu bitul r1 – rezervat – al câmpului de control).
- Câmpul de control este format din 6 biţi: r1 şi r0 sunt rezervaţi şi sunt transmişi dominanţi (în cazul cadrului extins, r1 devine IDE şi este transmis recesiv) şi un câmp de 4 biţi DLC0…DLC3 care definesc numărul de octeţi al mesajului din câmpul de date (*Data Field*). Considerând DLC0 ca cel mai puţin semnificativ bit şi identificând bitul recesiv ca 0 LOGIC, numărul de octeţi ai *Data Field* se determină prin convertirea în zecimal a câmpului. Valoarea maximă a câmpului este opt.
- Câmpul de date conţine cei până la opt octeţi de informaţie.
- Câmpul CRC conţine o secvenţă CRC şi un delimitator. Secvenţa CRC este determinată pentru secvenţa de biţi începând cu bitul de start. Delimitatorul constă într-un bit recesiv.
- Câmpul de confirmare este format din 2 biţi: ACK SLOT şi un delimitator. Toate dispozitivele care au recepţionat o secvenţă CRC corectă marchează aceasta prin înlocuirea bitului recesiv ACK SLOT trimis de emiţător printr-un bit dominant.
- Cadrele de date sau cerere de date sunt terminate cu un câmp sfârşit cadru care conţine 7 biţi recesivi.
- Indicatorul eroare poate fi de două tipuri: activ (format din 6 biţi dominanţi) sau pasiv (format din 6 biţi recesivi numai dacă vreun bit nu a fost suprascris de un alt nod). Indicatorul de eroare activ contrazice regula de adăugare a biţilor (maxim 6 biţi consecutivi de acelaşi fel), regulă aplicată de la start cadru până la delimitatorul CRC sau strică structura câmpurilor ACK sau sfârşit. În consecinţă, toate nodurile detectează eroarea şi fiecare în parte transmite un cadru corespunzător. Astfel, secvenţa de biţi dominanţi poate fi afectată prin suprapunerea mai multor

indicatori de eroare transmişi de nodurile individuale. Un nod pasiv care detectează o eroare va transmite un indicator pasiv de eroare. Nodul pasiv aşteaptă 6 biţi consecutivi de aceeaşi polaritate începând cu startul indicatorului de eroare. Delimitatorul de eroare constă într-o secvenţă de 8 biţi recesivi. După transmiterea unui indicator de eroare, fiecare nod emite biţi recesivi şi monitorizează linia până când detectează un bit recesiv. După aceasta mai emite 7 biţi recesivi.

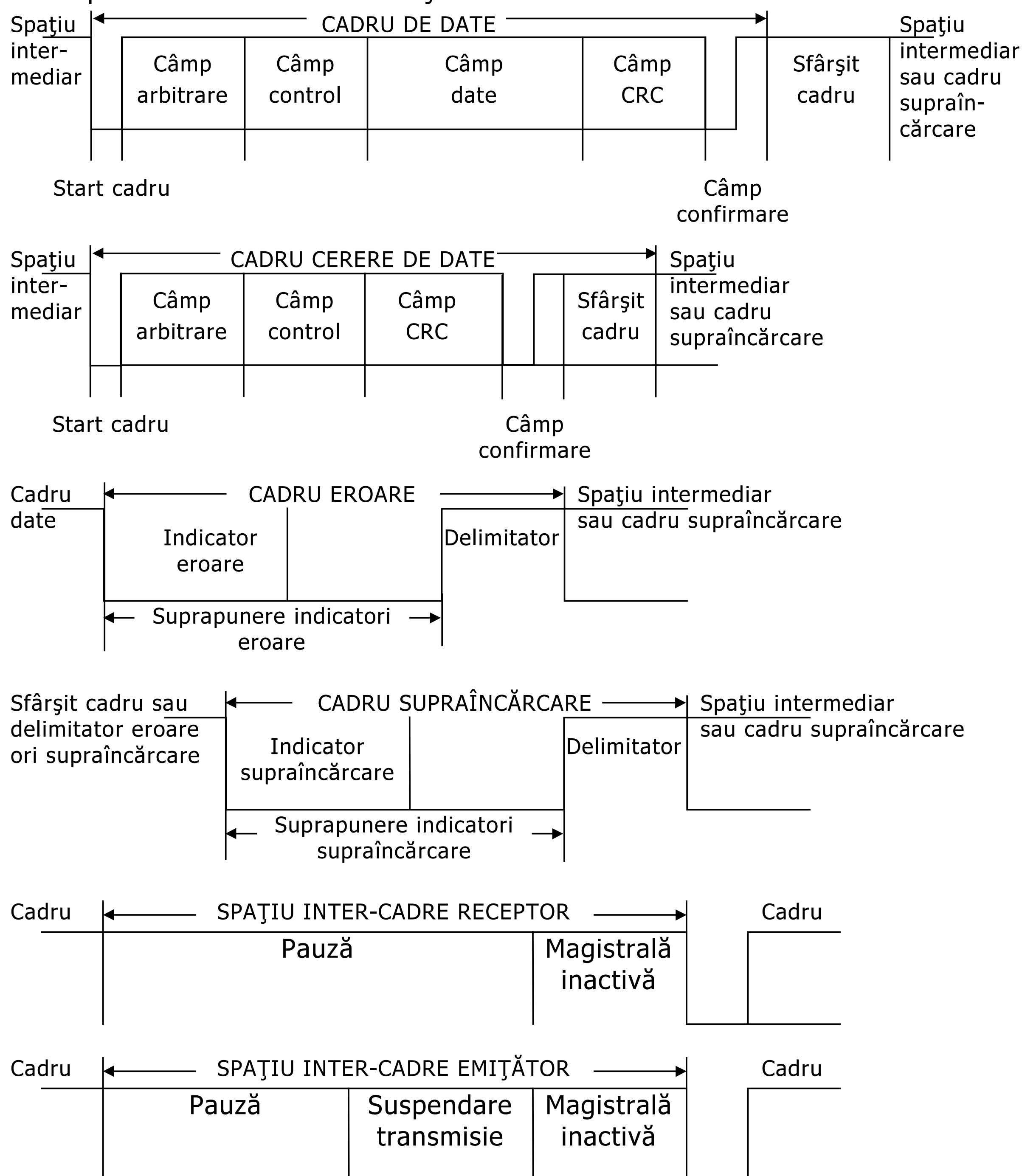


Figura 3.6. Tipurile de cadre de mesaje ale CAN

- Indicatorul supraîncărcare este transmis în câteva cazuri particulare:
  - receptoare care necesită o întârziere a noului cadru de date sau cerere de date;
  - detectarea unui bit dominant pe prima sau a doua poziţie a unei pauze;

- detectarea unui bit dominant pe ultima poziţie a unui delimitator de eroare sau supraîncărcare.

Indicatorul este format din 6 biţi dominanţi. Prezenţa sa distruge forma câmpului pauză toate nodurile detectând o condiţie de supraîncărcare şi iniţiind o transmisie a indicatorului supraîncărcare. Dacă în câmpul pauză este detectat un bit dominant pe poziţia a treia, acesta este considerat ca start cadru.

Delimitatorul de supraîncărcare este format din opt biţi, ca delimitatorul de eroare. După transmiterea indicatorului, nodurile monitorizează magistrala până când este detectată o trecere la dominant→recesiv. După aceasta, fiecare nod iniţiază transmiterea simultană a 7 biţi recesivi suplimentari.

- Spaţiile inter-cadre sunt folosite pentru separarea cadrelor de date şi cerere de date de cadrele următoare. Acest câmp de biţi este format din câmpurile pauză şi magistrală inactivă între care se intercalează, în situaţia unui nod pasiv la erori care a fost emiţător, câmpul suspendare transmisie. Câmpul pauză constă în 3 biţi recesivi. Câmpul magistrală inactivă poate avea o lungime arbitrară deoarece în această stare magistrala este considerată liberă, situaţie care va dura până când un nod va produce un bit dominant – start cadru.

  După ce un nod pasiv la erori a transmis un mesaj, el emite opt biţi recesivi după câmpul pauză înainte de a transmite un nou mesaj sau a elibera magistrala.

d) Prelucrarea erorilor

Magistrala CAN admite cinci tipuri de erori:

- Eroarea de bit este produsă atunci când un nod nu recepţionează de pe linie acelaşi bit transmis. În anumite cazuri particulare nu este generată această eroare.
- Eroarea de bit de adaus se generează dacă există 6 biţi egali în câmpurile start cadru, arbitrare, control, date şi secvenţa CRC.
- Eroarea CRC este semnalată dacă restul polinomului CRC calculat nu este identic cu codul recepţionat.
- Eroarea de formă este detectată dacă într-un câmp de formă fixă sunt detectaţi unul sau mai mulţi biţi incorecţi.
- Eroarea de confirmare este generată de un emiţător care nu monitorizează un bit dominant pe poziţia ACK SLOT.

e) Limitarea erorilor

Pentru a reduce numărul erorilor produse de un dispozitiv eventual defect, standardul CAN obligă un nod să fie în una din cele trei stări care urmează:

- eroare activă – nodul ia parte la comunicaţia pe linie trimiţând un indicator de eroare atunci când eroarea a fost detectată;

- eroare pasivă – nodul ia parte la comunicaţia pe linie trimiţând un indicator de eroare pasivă atunci când eroarea a fost detectată; astfel, după o emisie, un nod pasiv va aştepta înainte de iniţia transmisia următoare;
- nelegat la magistrală – nodul nu are nici o influenţă asupra liniei de comunicaţie.

Pentru limitarea erorilor, în fiecare dispozitiv sunt implementate două numărătoare: unul pentru erorile de transmisie, respectiv recepţie. Aceste numărătoare sunt incrementate respectând următoarele reguli:

a) când un receptor detectează o eroare, este incrementat contorul de erori la recepţie, cu excepţia cazului când a fost detectată o eroare de bit în timpul transmiterii unui indicator de eroare activă sau indicator de supraîncărcare;
b) în situaţia în care un receptor, după transmiterea unui indicator de eroare, detectează primul bit ca bit dominant, contorul de erori la recepţie este mărit cu 8;
c) când un emiţător trimite un indicator de eroare, contorul de erori la transmisie este mărit cu 8; ca excepţii la această regulă, când nu se modifică contorul, trebuie menţionate eroarea de confirmare a unui emiţător pasiv şi eroarea de bit de adaus survenită pe timpul arbitrării;
d) dacă un emiţător detectează o eroare de bit în timpul transmiterii unui indicator de eroare activă sau indicator de supraîncărcare, contorul de erori de transmisie este mărit cu 8;
e) dacă un receptor detectează o eroare de bit în timpul transmiterii unui indicator de eroare activă sau indicator de supraîncărcare, contorul de erori de recepţie este mărit cu 8;
f) orice nod tolerează până la 7 biţi consecutivi dominanţi după transmiterea unui indicator de eroare activă sau indicator de supraîncărcare. După detectarea a 14 biţi dominanţi consecutivi sau după detectarea a 8 biţi dominanţi consecutivi care urmează unui indicator de eroare pasivă, precum şi după fiecare secvenţă adiţională de 8 biţi consecutivi dominanţi, fiecare contor de transmisie şi recepţie sunt mărite cu 8;
g) după transmiterea reuşită a unui mesaj, contorul de erori la transmisie este decrementat (dar fără a căpăta valori negative);
h) după recepţia reuşită a unui mesaj contorul de erori la recepţie este decrementat (dacă a fost între 1 şi 127), rămâne nemodificat (dacă a fost 0) ori primeşte o valoare în domeniul 119...127 (dacă a fost mai mare de 127);
i) un nod este pasiv la erori în situaţia în care contoarele de erori la transmisie sau recepţie au valori mai mari de 128;
j) un nod pasiv devine activ dacă numărătoarele de erori la transmisie sau recepţie au valori mai mici de 127;

k) un nod este deconectat atunci când contorul de erori la transmisie este mai mare de 256;

l) unui nod deconectat îi este permis să redevină activ (cu ambele contoare şterse) după 128 de cazuri în care 11 biţi recesivi consecutivi au fost monitorizaţi pe magistrală.

Dacă la iniţializare este conectat la linie un singur nod, în situaţia în care acest nod va transmite mesaje nu va primi nici o confirmare, detectând o eroare care provoacă repetarea mesajului. El poate să devină pasiv la erori dar nu va fi deconectat.

f) Module CAN din microcontrolere

Marile firme europene au introdus în familiile lor de microcontrolere şi module pentru interfeţe CAN, atât pentru circuitele de 16 biţi cât şi pentru cele de 8 biţi (C167CR şi C164CI – realizat ca periferic X-BUS, respectiv C515C şi C505C de la firma Siemens, C592 şi C598 de la firma Philips etc.), circuite pentru conversie la standard CAN (PCA82C250), sau controlere pentru interfaţă CAN (SJA1000 etc.).

Circuitul 80C592

Microcontrolerul Philips 8xC592 este asemănător cu circuitul descris în capitolul 1, 8xC552, numai că la acesta, în locul interfeţei seriale I^2C (SIO1) este prezent o interfaţă CAN. Ca suport al modulului CAN, în acest circuit este prezent şi un modul DMA (*direct memory acces*) necesar pentru creşterea vitezei de transfer ale datelor între nucleul microcontrolerului şi interfaţa CAN.

Protocolul adoptat pentru acest circuit este CAN 2.0A (identificatori pe 11 biţi care asigură 2032 identificatori diferiţi).

Circuitul 8xC592 conţine toate circuitele necesare pentru logica de comunicare în reţeaua CAN. În exterior este necesar doar un transceiver, de exemplu PCA82C250.

Interfaţa între unitatea centrală şi CAN este făcută prin intermediul a patru registre speciale:

- CANADR: indică adresa unui registru din modulul CAN;
- CANDAT: registru de date;
- CANCON: registru de control funcţionare şi întreruperi;
- CANSTA: registru de stare şi indicatori DMA.

Zonele de adrese ale modului CAN constau în segmentul de control şi bufferele de date. Segmentul de control este programat pe durata iniţializării pentru a fi configuraţi parametrii de comunicare. Unitatea centrală poate modifica ulterior parametrii de comunicare pe baza acestui segment (este contraindicată modificarea registrelor "Cod acceptare" – ACR, "Mască acceptare" – AMR, "Sincronizare 0 şi 1" – BTR0 şi BTR1, "Control ieşire" – OCR). Mesajele de transmis sunt scrise în bufferul de emisie, în timp ce mesajele recepţionate corect pot fi citite din bufferul de recepţie.

Structura segmentului de control şi a bufferelor de emisie/recepţie sunt prezentate în figura 4.7.

Adresă		
00h	Control (CR)	Segment control
01h	Comandă (CRM)	
02h	Stare (SR)	
03h	Întrerupere (IR)	
04h	Cod acceptare (ACR)	
05h	Mască acceptare (AMR)	
06h	Sincronizare magistrală 0 (BTR0)	
07h	Sincronizare magistrală 1 (BTR1)	
08h	Control ieşire (OCR)	
09h	Test	

0Ah	Identificator (DSCR0)	Descriptor	Bufer emisie
0Bh	Bit RTR. Cod lungime mesaj (DSCR1)		
0Ch	Octet 1	Date	
0Dh	Octet 2		
0Eh	Octet 3		
0Fh	Octet 4		
10h	Octet 5		
11h	Octet 6		
12h	Octet 7		
13h	Octet 8		

14h	Identificator	Identificator	Descriptor	Bufer recepţie 0 sau 1
15h	Bit RTR. Cod lungime	Bit RTR. Cod lungime		
16h	Octet 1	Octet 1	Date	
17h	Octet 2	Octet 2		
18h	Octet 3	Octet 3		
19h	Octet 4	Octet 4		
1Ah	Octet 5	Octet 5		
1Bh	Octet 6	Octet 6		
1Ch	Octet 7	Octet 7		
1Dh	Octet 8	Octet 8		

Figura 3.7. Adresele interne ale modulului CAN din 8xC592

Registrul de control (CR)

Conţinutul registrului CR este folosit pentru schimbarea caracteristicilor modulului CAN.

Structura registrului CR este prezentată în tabelul 4.2.

Tabelul 3.2

CR (0)	TM	S	RA	OIE	EIE	TIE	RIE	RR
TM	Mod test. Este destinat testărilor la fabricant. 0: modulul CAN funcţionează normal; 1: modulul CAN funcţionează în mod test.							
S	Bit sincronizare. Modificarea biţilor S şi RA este posibilă numai dacă bitul RR este setat. După o iniţializare externă S este nedefinit. 0: tranziţiile magistralei recesiv→dominant sunt folosite pentru resincronizare; 1: sunt folosite pentru resincronizare ambele tipuri de tranziţii.							
RA	Referinţă activă. După o iniţializare externă RA este setat. 0: se foloseşte ca referinţă o tensiune externă; 1: se foloseşte ca referinţă tensiunea ½ AV_{DD}.							

OIE	Validare întrerupere depăşire. 0: nu sunt generate întreruperi la depăşirile de date; 1: dacă bitul depăşire date este setat, este generată o întrerupere.
EIE	Validare întreruperi erori magistrală. 0: nu sunt generate întreruperi pentru erorile de magistrală; 1: dacă bitul eroare magistrală este setat, este generată o întrerupere.
TIE	Validare întrerupere transmisie. 0: nu sunt generate întreruperi la terminarea unei transmisii de date; 1: este generată o întrerupere după ce un mesaj a fost transmis şi bufferul de emisie este disponibil.
RIE	Validare întrerupere recepţie. 0: nu sunt generate întreruperi recepţie; 1: este generată o întrerupere după ce un mesaj a fost recepţionat fără erori.
RR	Cerere iniţializare. Pe durata unei iniţializări externe sau când bitul de stare a magistralei este setat, logica de control a interfeței (IML) setează acest bit. După ce bitul este şters, modulul CAN aşteaptă: a) un semnal magistrală liberă (11 biţi recesivi), dacă iniţializarea precedentă a fost generată de o iniţializare externă sau un reset unitate centrală; b) 128 de semnale magistrală liberă, dacă iniţializarea precedentă a fost generată de modulul CAN; c) dacă RR este setat din orice motiv, biţii control, stare şi întrerupere sunt afectaţi (conform tabelului 4.3). Registrele de la adresele 4…8 sunt accesibile numai dacă RR este setat. 0: tranziţia 1→0 a RR are ca efect funcţionarea normală a modului CAN; 1: modulul CAN elimină mesajul curent intrând în starea de iniţializare sincronă cu ceasul sistem.

Tabelul 3.3

Tip	Bit	Simbol	Funcţie	Efect
Control	CR.7	TM	Mod test	Dezactivat
	CR.5	RA	Referinţă activă	Setat (ieşire)
Comandă	CMR.7	RX0A	RX0 activ	Setat (RX0 = CRX0)
	CMR.6	RX1A	RX1 activ	Setat (RX1 = CRX1)
	CMR.4	SLP	Inactiv	Şters (atenţionare)
	CMR.3	COS	Ştergere stare depăşire	Setat (şters)
	CMR.2	RRB	Eliberare buffer recepţie	Setat
	CMR.1	AT	Suprimare transmisie	Şters
	CMR.0	TR	Cerere transmisie	Şters
Stare	SR.7	BS	Stare magistrală	Şters (magistrală activă)
	SR.6	ES	Stare erori	Şters (nu există erori)
	SR.5	TS	Stare emisie	Şters (inactiv)
	SR.4	RS	Stare recepţie	Şters (inactiv)
	SR.3	TCS	Emisie completă	Setat (completă)
	SR.2	TBS	Acces buffer transmisie	Setat (liber)
	SR.1	DO	Depăşire date	Şters (inexistent)
	SR.0	RBS	Stare buffer recepţie	Şters (registru descărcat)
Întreruperi	IR.3	OI	Întrerupere depăşire	Şters
	IR.1	TI	Întrerupere transmisie	Şters
	IR.0	RI	Întrerupere recepţie	Şters

Registrul de comandă (CMR)

Structura şi funcţiunile biţilor registrului de comandă sunt descrise în tabelul 4.4.

<table>
<tr><td colspan="9" align="right">Tabelul 3.4</td></tr>
<tr><td>CMR (1)</td><td>RX0A</td><td>RX1A</td><td>WUM</td><td>SLP</td><td>COS</td><td>RRB</td><td>AT</td><td>TR</td></tr>
</table>

<table>
<tr><td>RX0A
RX1A</td><td>Reflectă starea indicatorilor respectivi (figura 4.8). Se recomandă să fie schimbaţi numai pe durata stării de iniţializare (RR=1 LOGIC)
<table>
<tr><td colspan="2">Control</td><td rowspan="2">RX0</td><td rowspan="2">RX1</td></tr>
<tr><td>RX0A</td><td>RX1A</td></tr>
<tr><td>1</td><td>1</td><td>CRX0</td><td>CRX1</td></tr>
<tr><td>1</td><td>0</td><td>CRX0</td><td>½ AV_{DD}</td></tr>
<tr><td>0</td><td>1</td><td>½ AV_{DD}</td><td>CRX1</td></tr>
<tr><td>0</td><td>0</td><td colspan="2">Inactiv</td></tr>
</table>
</td></tr>
<tr><td>WUM</td><td>Mod atenţionare. Se setează concomitent cu bitul SLP.
0: diferenţele dintre semnalele RX sunt folosite pentru atenţionare;
1: pentru atenţionare sunt folosite diferenţele dintre semnalele RX şi ½AV_{DD}. Recomandat pentru medii de transmisie cu interferenţe mari.</td></tr>
<tr><td>SLP</td><td>Inactiv. Modulul CAN intră în modul inactiv (adormit) dacă bitul SLP este setat, nu există nici o activitate pe magistrală şi nu există nici o întrerupere. După aceasta, modulul CAN este atenţionat dacă există activitate pe magistrală sau bitul SLP este şters. La atenţionare este generată o întrerupere specifică. După activare, dispozitivul CAN nu este capabil să recepţioneze un mesaj până când nu detectează un semnal "magistrală liberă".
0: controlerul funcţionează normal;
1: controlerul CAN este inactiv.</td></tr>
<tr><td>COS</td><td>Ştergere stare depăşire. Acest bit este folosit pentru confirmarea depăşirii semnalate de bitul "depăşire date". Comanda este executată numai după eliberarea ambelor registre de recepţie. Bitul trebuie setat simultan cu RRB.</td></tr>
<tr><td>RRB</td><td>Eliberare buffere recepţie. După citirea registrelor recepţie (RBF0 sau RBF1) unitatea centrală eliberează bufferul prin setarea acestui bit. Aceasta are ca rezultat posibilitatea ca un nou mesaj să devină imediat disponibil. Pentru a asigura executarea unei singure comenzi RRB, timpul minim între două astfel de comenzi este de 3 ceasuri sistem.</td></tr>
<tr><td>AT</td><td>Suprimare transmisie. AT este folosit pentru întreruperea imediată a unei transmisii solicitate anterior, de exemplu, pentru a transmite un mesaj urgent. O transmisie aflată în curs nu este întreruptă. Pentru a determina dacă mesajul iniţial a fost transmis, trebuie verificat bitul TCS.</td></tr>
<tr><td>TR</td><td>Cerere transmisie. Dacă bitul TR a fost setat de o comandă anterioară, cererea nu poate fi eliminată prin ştergerea bitului TR. Anularea cererii se poate face numai prin setarea bitului AT.</td></tr>
</table>

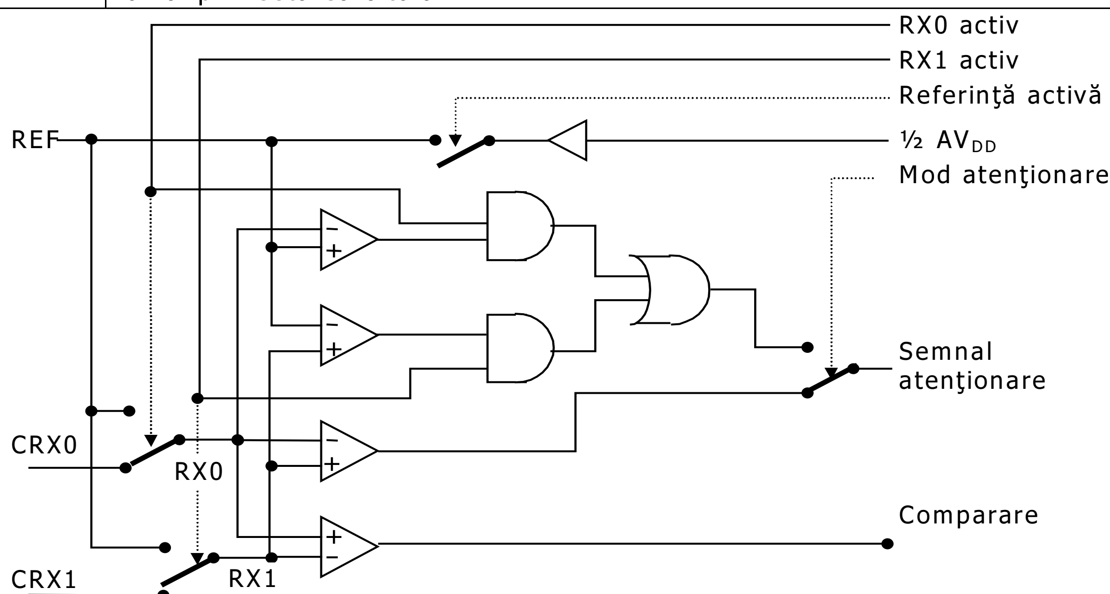


Figura 3.8. Configurarea receptorului CAN

Registrul de stare (SR)

Registrul de stare reflectă situaţia modului CAN şi este descris în tabelul 4.5.

Tabelul 3.5

SR (2)	BS	ES	TS	RS	TCS	TBS	DO	RBS

BS	Stare magistrală. Când magistrala este inactivă, modulul CAN va seta bitul RB. Modulul rămâne în această stare până când RR este şters. După aceasta, CAN aşteaptă timpul minim definit de protocol (128 de semnale "magistrală liberă") înainte de a activa magistrala, şterge bitul de stare eroare şi reiniţializa contorul de erori. 0: modulul este implicat în lucrul pe magistrala CAN; 1: modulul nu este implicat în lucrul pe magistrală.
ES	Stare eroare. 0: ambele numărătoare de erori nu au depăşit numărul maxim; 1: cel puţin un numărător de erori a depăşit numărul maxim.
TS	Stare transmisie. 0: nu este transmis nici un mesaj; 1: mesaj în curs de transmitere.
RS	Stare recepţie. Dacă atât RS cât şi TS sunt 0 LOGIC, magistrala CAN este inactivă. 0: nu este recepţionat nici un mesaj; 1: mesaj în curs de recepţionare.
TCS	Completare transmisie. TCS este şters ori de câte ori indicatorul TR este setat. Dacă un mesaj care a fost cerut şi apoi anulat nu a fost transmis, TCS rămâne 0 LOGIC. 0: mesajul solicitat anterior nu a fost transmis; 1: mesajul solicitat anterior a fost transmis.
TBS	Acces buffer transmisie. Dacă unitatea centrală scrie în bufferul de transmisie cât timp indicatorul TBA este şters, octetul scris este pierdut fără a se semnala. 0: unitatea centrală nu poate scrie în bufferul de transmisie; un mesaj fie aşteaptă să fie transmis, fie este în curs de transmitere; 1: unitatea centrală poate scrie în buffer.
DO	Depăşire date. Dacă este detectat DO=1, mesajul recepţionat este eliminat. Un mesaj admis pentru transmisie este memorat de asemenea într-un buffer de recepţie. Dacă modulul pierde arbitrarea, poate deveni receptor şi neavând buffer de recepţie disponibil se semnalează o eroare DO. DO nu provoacă transmisia unui cadru de depăşire. 0: nu a survenit nici o depăşire de la ultima comandă "Ştergere depăşiri"; 1: ambele buffere de recepţie sunt pline şi nu mai poate fi memorat primul octet al unui nou mesaj.
RBS	Stare buffer recepţie. Dacă indicatorul RRB este setat de unitatea centrală, RBS este şters de logica de control a interfeţei (IML). Dacă un nou mesaj este memorat într-unul din bufferele de recepţie, RBS este setat. 0: nu a survenit nici un mesaj de la ultima comandă RRB; 1: un nou mesaj este disponibil.

Registrul de întreruperi (IR)

Registrul de întreruperi permite identificarea sursei unei întreruperi. Dacă unul sau mai mulţi biţi ai acestui registru sun setaţi, este generată o întrerupere SIO1. Toţi biţii sunt şterşi de controlerul CAN după citirea registrului. Registrul IR este descris în tabelul 4.6.

Tabelul 3.6

IR (3)	–	–	–	WUI	OI	EI	TI	RI

WUI	Întrerupere atenţionare. 0: registrul a fost citit de unitatea centrală; 1: modul inactiv a fost abandonat.
OI	Întrerupere depăşire. Este setat sincron cu indicatorul "Depăşire date". 0: registrul a fost citit de unitatea centrală; 1: ambele registre recepţie conţin un mesaj, se primeşte un nou mesaj care nu poate fi păstrat şi indicatorul OIE (validare întrerupere depăşire) este setat.
EI	Întrerupere la eroare. 0: registrul a fost citit de unitatea centrală; 1: schimbarea indicatorilor ES sau BS, dacă EIE (validare întrerupere eroare) este setat.
TI	Întrerupere transmisie. 0: registrul a fost citit de unitatea centrală; 1: schimbarea indicatorului TBA, dacă TIE (validare întrerupere emisie) este setat.
RI	Întrerupere recepţie. RI şi RBS sunt setaţi concomitent. 0: registrul a fost citit de unitatea centrală; 1: un nou mesaj este disponibil în bufferul de recepţie şi RIE (validare întrerupere recepţie) este setat.

Registrul cod de acceptare (ACR)

Registrul ACR este o componentă a filtrului de acceptare a mesajelor a modulului CAN. Acest registru poate fi accesat dacă indicatorul RR este setat.

Dacă un mesaj recepţionat trece testul de acceptare şi este disponibil un buffer de recepţie, câmpurile control şi date sunt memorate în buffer; dacă nu există buffer liber este setat indicatorul "Depăşire date".

Pe durata transmisiei unui mesaj care a trecut testul de acceptare, mesajul este scris şi în propriul buffer de recepţie deoarece nu se poate şti dacă modulul va pierde arbitrarea şi va deveni receptor al mesajului. Dacă nu există buffer de recepţie liber este setat indicatorul "Depăşire date".

Structura registrului cod de acceptare este prezentată în tabelul 4.7.

Tabelul 3.7

ACR (4)	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0

AC7-AC0	Cod acceptare. Biţii AC7…AC0 primii opt biţi mai semnificativi ai identificatorului (ID10…ID3) trebuie să fie egali cu acei biţi de poziţie care sunt marcaţi relevanţii de Registrul mască de acceptare (AMR). Mesajul este acceptat dacă este satisfăcută următoarea ecuaţie: (ID10…ID3)=[(AC7…AC0)+(AM7…AM0)]=1111 1111_B

Registrul mască de acceptare (AMR)

Registrul AMR este o parte a filtrului de acceptare a modulului CAN. Acest registru poate fi accesat dacă indicatorul RR este setat. Registrul AMR determină care din biţii ACR sunt relevanţi pentru filtrare. Structura registrului este prezentată în tabelul 4.8.

Tabelul 3.8

AMR (5)	AM7	AM6	AM5	AM4	AM3	AM2	AM1	AM0

AM7-AM0	Mască de acceptare. 0: biţii din această poziţie sunt relevanţi pentru filtrare; 1: biţii din această poziţie nu au importanţă pentru filtrare.

Registrul 0 de sincronizare a magistralei (BTR0)

Conţinutul registrului BTR0 defineşte valorile prescalerului pentru ratele de transmisie (BRP), precum şi lăţimea salturilor de sincronizare (SJV). Acest registru poate fi accesat dacă indicatorul RR este setat. Structura registrului BTR0 este prezentată în tabelul 4.9.

Tabelul 3.9

BTR0 (6)	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
SJW	Pentru a compensa diferenţele de fază între diferitele oscilatoare de pe magistrală, fiecare controler de magistrală trebuie să se resincronizeze cu orice semnal relevant al mesajului curent. SJW defineşte numărul maxim de tacte cu care o perioadă de bit poate fi scurtată sau lungită la resincronizare: $t_{SJW} = t_{SCL}(2SJW1+SJW0+1)$							
BRP	Perioada ceasului de serializare este programabilă şi se poate determina cu relaţia: $t_{SCL} = 2t_{CLK}(32BRP5+16BRP4+8BRP3+4BRP2+2BRP1+BRP0+1)$; t_{CLK} = perioada microcontrolerului.							

Registrul 1 de sincronizare a magistralei (BTR1)

Conţinutul registrului BTR1 defineşte mărimea perioadei unui bit, poziţionarea punctelor de eşantionare şi numărul de eşantioane din fiecare punct. Acest registru poate fi accesat dacă indicatorul RR este setat. Structura registrului BTR1 este prezentată în tabelul 4.10.

Tabelul 3.10

BTR1 (7)	SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0
SAM	Numărul de eşantioane: 0: 1 eşantion; recomandat pentru magistralele rapide; 1: 3 eşantioane; recomandat pentru magistrale mai lente, unde pot fi prezente interferenţe puternice.							
TSEG2 TSEG1	Determină numărul de cicluri pe perioada de bit şi poziţia punctului de eşantionare. Poziţionarea corectă a punctului de eşantionare este esenţială pentru funcţionarea corectă a transmisiei. Trebuie avute în considerare următoarele: • “Start cadru” implică o sincronizare hardware pe prima tranziţie recesiv-dominant. Pe durata arbitrării, unele controlere CAN pot emite simultan; acest motiv poate implica luarea în calcul a unui timp de propagare dublu. • Pentru a evita eşantionarea pe o poziţie incorectă, este necesară introducerea unor buffere de sincronizare de ambele părţi ale punctului de eşantionare. TSEG1 asigură compensarea propagării întârzierilor şi sincronizarea înainte de punctul de eşantionare. Se determină cu relaţia: $t_{TSEG1} = t_{SCL}(8TSEG1.3+4TSEG1.2+2TSEG1.1+TSEG1.0+1)$ TSEG2 asigură întârzieri suplimentare necesare calculului următoarelor nivele ale biţilor, precum şi sincronizarea după punctul de eşantionare. Se determină cu relaţia: $t_{TSEG2} = t_{SCL}(4TSEG2.2+2TSEG2.1+TSEG2.0+1)$							

Registrul de control a ieşirii (OCR)

Registrul OCR permite, sub controlul programului, setarea unor configuraţii de ieşire variate. Acest registru poate fi accesat dacă indicatorul RR este setat. Dacă modulul este inactiv (*sleep*), pe pinii CTX0 şi CTX1 este scos un nivel ‘recesiv’. Dacă modulul este în stare de iniţializare (RR=1), amplificatoarele de ieşire sunt flotante.

Structura registrului OCR este prezentată în tabelul 4.11.

Tabelul 3.11

OCR (8)	OCTP1	OCTN1	OCPOL1	OCTP0	OCTN0	OCPOL0	OCMODE1	OCMODE0

OCTP1	Comandă tranzistorul de ieşire CTX1 legat la V_{DD}.
OCTN1	Comandă tranzistorul de ieşire CTX1 legat la V_{SS}.
OCPOL1	Comandă polaritatea semnalului CTX1.
OCTP0	Comandă tranzistorul de ieşire CTX0 legat la V_{DD}.
OCTN0	Comandă tranzistorul de ieşire CTX0 legat la V_{SS}.
OCPOL0	Comandă polaritatea semnalului CTX0.

Amplificator	OCTPx	OCTNx	OCPOLx	TXD	CTXx
Flotant	0	0	0	0	flotant
	0	0	0	1	flotant
	0	0	1	0	flotant
	0	0	1	1	flotant
Pull-down	0	1	0	0	dominant (0)
	0	1	0	1	flotant
	0	1	1	0	flotant
	0	1	1	1	dominant (0)
Pull-up	1	0	0	0	flotant
	1	0	0	1	recesiv (1)
	1	0	1	0	recesiv (1)
	1	0	1	1	flotant
Push/Pull	1	1	0	0	dominant (0)
	1	1	0	1	recesiv (1)
	1	1	1	0	recesiv (1)
	1	1	1	1	dominant (0)

OCMODE1 OCMODE0	Comandă modurile de ieşire.
	00: Mod ieşire Bi-phase. Spre deosebire de modul normal, reprezentarea biţilor este variabilă în timp. Dacă modulul este decuplat galvanic de linia de transmisie, semnalele nu trebuie să conţină componente DC. Pe durata biţilor recesivi, toate ieşirile sunt flotante; biţii dominanţi sunt trimişi alternativ pe CTX0 şi CTX1.
	01: Mod test. Pentru pinul CTX0 funcţionarea este identică cu modul normal. Testul este folosit numai la producător pentru a măsura timpii de întârziere.
	10: Mod normal. Secvenţele de biţi (TXD) sunt transmise prin intermediul CTX0 şi CTX1. TXD este data care urmează să fie transmisă. Nivelul tensiunii pe CTX0 şi CTX1 depinde de valorile programate prin OCTPx, OCTNx (flotant, pull-up, pull-down, push-pull) şi OCPOLx.
	11: Mod ceas. Pentru pinul CTX0 funcţionarea este identică cu modul normal. Datele pe pinul CTX1 sunt înlocuite de tactul de transmisie serială. Frontul crescător al tactului marchează începutul unui bit. Perioada tactului este t_{SCL}.

Bufferul de transmisie (DSCR1, DSCR0 şi câmpurile de date)

Structura registrelor descriptor şi câmpului de date este prezentată în tabelul 4.12.

Tabelul 3.12

DSCR1 (10)	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
DSCR0 (11)	ID2	ID1	ID0	RTR	DLC3	DLC2	DLC1	DLC0

ID10…ID0	Identificatorul. Cei 11 biţi sunt trimişi pe magistrală în timpul procesului de arbitrare, cea mai mică valoare binară desemnând prioritatea cea mai mare. De asemenea, identificatorul serveşte ca nume al mesajelor fiind folosit la filtrare.
RTR	Cerere de date: 0: va fi transmis un cadru de date; 1: va fi transmis un cadru cerere de date.

<table>
<tr><td colspan="2">DLC3…0</td><td colspan="8">Cod lungime mesaj. Numărul de octeţi din câmpul de date este codificat de acest câmp. La iniţierea transmisiei unui cadru cerere de date, DLC nu este luat în considerare, fiind forţat la 0 LOGIC. Cu toate acestea, pentru mesajele standard, DLC trebuie specificat corect, cu valori de la 0 la 8, conform cu relaţia:
$$\text{Număr octeţi} = 8DLC3+4DLC2+2DLC1+DLC0$$</td></tr>
<tr><td colspan="2">DATE (12…19)</td><td>DATA7</td><td>DATA6</td><td>DATA5</td><td>DATA4</td><td>DATA3</td><td>DATA2</td><td>DATA1</td><td>DATA0</td></tr>
<tr><td colspan="2">DATA7…0</td><td colspan="8">Reprezintă octeţii de date care formează mesajul (maxim 8 octeţi).</td></tr>
</table>

Bufferul de recepţie (DSCR1, DSCR0 şi câmpurile de date)

Structura şi semnificaţia octeţilor bufferului de recepţie este identică cu a celui de transmisie, cu excepţia adresei registrelor interne care este în domeniul 20…29.

Registrele speciale pentru interfaţare cu unitatea centrală

Prin intermediul a patru registre speciale, CANADR, CANDAT, CANCON şi CANSTA, unitatea centrală are una acces deplin asupra modulului CAN şi DMA. Structura şi semnificaţia registrelor este prezentată în tabelul 4.13.

<table>
<tr><td colspan="10" align="right">Tabelul 3.13</td></tr>
<tr><td colspan="2">CANADR (DBh)</td><td>DMA</td><td>–</td><td>AutoInc</td><td>CANA4</td><td>CANA3</td><td>CANA2</td><td>CANA1</td><td>CANA0</td></tr>
<tr><td colspan="2">DMA</td><td colspan="8">Control logică DMA. Această logică permite transferul unui mesaj complet (10 octeţi) între modulul CAN şi memoria RAM internă în maxim 2 instrucţiuni. Un transfer DMA este stabilit mai întâi prin scrierea adresei de RAM (00h la FFh) în CANSTA iar apoi setarea simultană a indicatorilor DMA şi adreselor bufferelor Tx sau Rx din CANADR; adresa RAM indică locaţia primului octet care va fi transferat. Setarea DMA produce evaluarea automată a DLC şi apoi iniţiază transferul.
Pentru a iniţia un transfer TX-DMA în CANADR trebuie scris 8Ah. Apoi mesajul complet (2 octeţi descriptori şi 0…8 octeţi date) de la adresa RAM sunt transferate în bufferul de transmisie.
Transferul RX-DMA se face scriind în CANADR o valoare de la 94h la 9Dh, pentru a selecta din mesaj octeţii doriţi (toţi opt, respectiv nici unul).
După un transfer DMA reuşit, indicatorul DMA este şters.</td></tr>
<tr><td colspan="2">AutoInc</td><td colspan="8">Dacă AutoInc este setat, conţinutul CANADR este incrementat automat după orice acces la registrul CANDAT. Incrementarea CANADR peste valoarea 3Fh şterge indicatorul AutoInc şi registrul CANADR.</td></tr>
<tr><td colspan="2">CANA4…0</td><td colspan="8">Definesc adresa registrului intern din modulul CAN care va fi accesat prin intermediul CANDAT.</td></tr>
<tr><td colspan="2">CANDAT (DAh)</td><td>CAND7</td><td>CAND6</td><td>CAND5</td><td>CAND4</td><td>CAND3</td><td>CAND2</td><td>CAND1</td><td>CAND0</td></tr>
<tr><td colspan="2">CAND7…0</td><td colspan="8">Registrul CANDAT apare ca un port către registrele interne ale modulului CAN selectate de CANADR. Scrierea sau citirea CANDAT este de fapt un acces la registrul intern CAN adresat de CANADR.</td></tr>
<tr><td rowspan="2">CANCON (D9h)</td><td>R</td><td>–</td><td>–</td><td>–</td><td>WUI</td><td>OI</td><td>EI</td><td>TI</td><td>RI</td></tr>
<tr><td>W</td><td>RX0A</td><td>RX1A</td><td>WUM</td><td>SLP</td><td>COS</td><td>RRB</td><td>AT</td><td>TR</td></tr>
<tr><td colspan="10">Structura indicatorilor este identică cu cea descrisă la registrele IR (pentru citire), respectiv CMR (pentru scriere)</td></tr>
<tr><td rowspan="2">CANSTA (DFh…D8h)</td><td>R</td><td>BS</td><td>ES</td><td>TS</td><td>RS</td><td>TCS</td><td>TBS</td><td>DO</td><td>RBS</td></tr>
<tr><td>W</td><td>RAMA7</td><td>RAMA6</td><td>RAMA5</td><td>RAMA4</td><td>RAMA3</td><td>RAMA2</td><td>RAMA1</td><td>RAMA0</td></tr>
<tr><td colspan="10">Structura indicatorilor este identică cu cea descrisă la registrele RS (pentru citire). Scrierea în CANSTA setează valorile RAMA7…0 corespunzător adresei din memoria RAM internă care va fi folosită pentru un transfer DMA.</td></tr>
</table>

Conectarea microcontrolerului 8xC592 la magistrala CAN

Interfaţarea între microcontroler şi linia de transmisie se face prin intermediul unui transceiver. Dispozitivul are următoarele funcţiuni:

- converteşte semnalele CTX0 şi CTX1 în nivele de tensiuni compatibile cu linia de transmisie;
- converteşte nivelele de tensiune de pe linie în semnale compatibile cu intrările CRX0 şi CRX1.

Conectarea fizică între controler şi linie este specifică aplicaţiei. Funcţie de cerinţele transferului de date, transceiverul poate fi realizat mai simplu sau mai complicat, cea mai ieftină soluţie constând în câteva rezistenţe iar cea mai complexă în câteva componente externe şi chiar câteva circuite integrate.

Condiţiile impuse transceiverului pot fi separate în două categorii, pentru emiţător şi pentru receptor.

Liniile emiţătorului (CTX0 şi CTX1) pot fi programate individual prin intermediul registrului OCR. Astfel, emiţătorul este uşor de proiectat pentru orice fel de linie diferenţială.

Receptorul constă într-un comparator diferenţial între liniile CRX0 şi CRX1. Cu excepţia comparării diferenţiale ale semnalelor de pe linie, referinţa comparatorului poate fi comutată de pe unul din semnale pe tensiunea de referinţă a modulului V_{REF}.

O schemă completă de transceiver, cu izolare galvanică între modulul CAN şi linia de transmisie, este prezentată în figura 4.9.

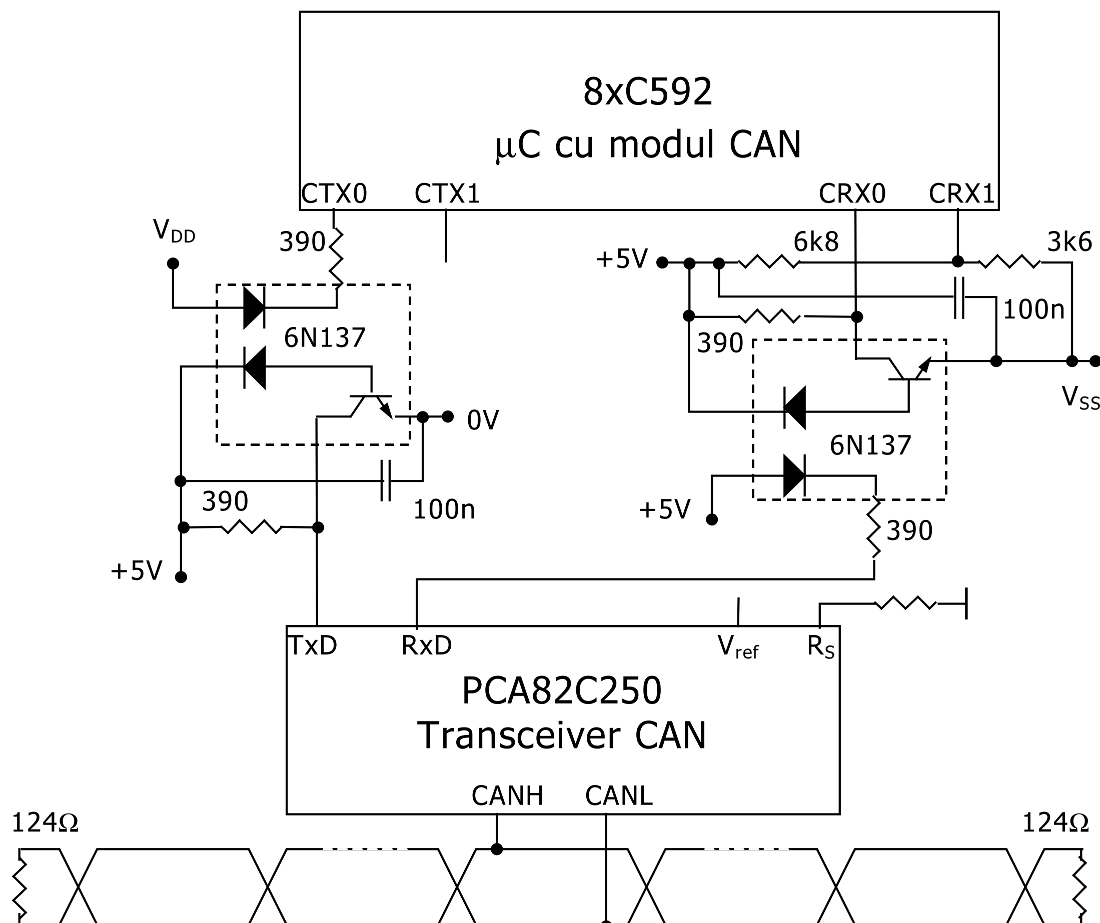


Figura 3.9. Interfaţarea CAN cu linia de transmisie

Anexe

Definiţiile constantelor sistemului de dezvoltare, adresele porturilor şi a perifericelor din fişierul SYSTEM.H.

```
/***************************************************************************\
**  Titlu:        SYSTEM.H                                               **
**  Descriere:    Declaratiile constantelor sistemului                    **
**                                                                        **
**  Versiune:     2.0                                                     **
**  Inceut la:    August 95                                               **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania      **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                      **
**                                                                        **
**  Acest modul contine urmatoarele functii:                              **
**                declaratii de constatnet si macrouri pentru:            **
**                - RTC, LCD, I2C, EEPROM, porturi, tastatura,            **
**                  controller intrerupri, wachdog                        **
**                                                                        **
**  Ultima modificare la data: 23 nov 1998                                **
**  Istoric:                                                              **
**                                                                        **
**  Observatii:                                                           **
**                                                                        **
\***************************************************************************/
#ifndef _SYSTEM_H_
#define _SYSTEM_H_
#include <reg552.h>

#define EX_WATCHDOG()   T2 = T2 ? 0 : 1         /* watchdog extern ADM691 */

/* RTC ==============================================================  */
#define RTC_ADR         0x2800                  /* Adrsa circuitului RTC */
#define regSEC1         (RTC_ADR | 0x0000)      /* = registru 0 din RTC */
#define regSEC10        (RTC_ADR | 0x0001)      /* = registru 1 din RTC */
#define regMIN1         (RTC_ADR | 0x0002)      /* = registru 2 din RTC */
#define regMIN10        (RTC_ADR | 0x0003)      /* = registru 3 din RTC */
#define regHOUR1        (RTC_ADR | 0x0004)      /* = registru 4 din RTC */
#define regHOUR10       (RTC_ADR | 0x0005)      /* = registru 5 din RTC */
#define regDAY1         (RTC_ADR | 0x0006)      /* = registru 6 din RTC */
#define regDAY10        (RTC_ADR | 0x0007)      /* = registru 7 din RTC */
#define regMON1         (RTC_ADR | 0x0008)      /* = registru 8 din RTC */
#define regMON10        (RTC_ADR | 0x0009)      /* = registru 9 din RTC */
#define regYEAR1        (RTC_ADR | 0x000a)      /* = registru A din RTC */
#define regYEAR10       (RTC_ADR | 0x000b)      /* = registru B din RTC */
#define regWEEK         (RTC_ADR | 0x000c)      /* = registru C din RTC */
#define regD_RTC        (RTC_ADR | 0x000d)      /* = registru D din RTC */
#define regE_RTC        (RTC_ADR | 0x000e)      /* = registru E din RTC */
#define regF_RTC        (RTC_ADR | 0x000f)      /* = registru F din RTC */

extern  xdata char      RTC_DATA[21];
#define RTC_DATE        RTC_DATA + 2
#define RTC_TIME        RTC_DATA + 11
#define RTC_WEEK        RTC_DATA[20]
#define RTC_WEEK_STR    RTC_DATA

/* LCCD ============================================================= */
#define LCD_ADR         0x0100
#define LCDRS           0x0001
#define LCDCMD          LCDadr
#define LCDDATA         LCDadr | LCDRS
#define LCDwcmd(c)      XBYTE[LCD_ADR + 0] = c
#define LCDwdta(d)      XBYTE[LCD_ADR + 1] = d
#define LCDrstate()     XBYTE[LCD_ADR + 2]
#define LCDrdta()       XBYTE[LCD_ADR + 3]
#define waitLCD()       while(LCDrstate() & 0x80)
#define LCD_SIZE        16
```

```
/* ADRESE I2C ============================================================ */
#define EEPROM          0xa0    /* adreaa EEPROM */
#define REL1            0x40    /* Bank 0 de relee - iesire */
#define REL2            0x42    /* Bank 1 de relee - iesire */
#define REL3            0x44    /* Bank 2 de relee - iesire */
#define REL4            0x46    /* Bank 3 de relee - iesire */
#define REL5            0x48    /* Bank 4 de relee - iesire */
#define REL6            0x4a    /* Bank 5 de relee - iesire */
#define REL7            0x4c    /* Bank 6 de relee - iesire */
#define REL8            0x4e    /* Bank 7 de relee - iesire */

/* EEPROM ================================================================ */
#define _E2P_MAX        512     /* capacitatea EEPROM-ului in octeti */
#define _E2P_BUF        512

/* CONTROLERUL DE INTRERUPERI ============================================ */
#define INTX_CTR        0x3000  /* Adresa controlerului de intrerupere */
#define INTX_LINE       (XBYTE[INTX_CTR] & 0xf) /* lina care a generat intr. */

/* IESIRI DIGITALE (CU LATCH) ============================================ */
#define DO1             0x0800
#define DO2             0x1000

/* INTRARI/IESIRI DIGITALE (FARA LATCH) ================================== */
#define IO1             0x1800
#define IO2             0x2000
#define ODE             0x3800

/* TASTAURA ============================================================== */
#define KBRD_BUFF_SIZE  4
#define KLOCK           ((P5 & 0x20) ? 0 : 1)

#endif
```

Definiţiile unor tipuri de date des utilizate în programele prezentate în fişierul TYPEDEF.H.

```
/**************************************************************************\
**  Titlu:        TYPEDEF.H                                              **
**  Descriere:    Declaratiile variabile byte, word, lword               **
**                                                                       **
**  Versiune:     2.0                                                    **
**  Inceut la:    August 95                                              **
**  Autor:        Stefan Suceveanu, Pratco s.r.l. Bucuresti, Romania     **
**  Compilator C: C51 V2.27, Franklin Software, Inc.                     **
**                                                                       **
**  Acest modul contine urmatoarele functii:                             **
**                                                                       **
**  Ultima modificare la data: 23 nov 1998                               **
**  Observatii:                                                          **
**                                                                       **
\**************************************************************************/
#ifndef _TYPEDEF_H
#define _TYPEDEF_H      1

#define byte unsigned char
#define word unsigned int
#define lword unsigned long

// citirea partii high si low a unei varibile pe 16 biti
#define HIGH(X)         ((byte)(*( (byte*)(&(X)) )))
#define LOW(X)          ((byte)(*(((byte*)(&(X))) + 1)))

// adresare partii high si low a unei varibile pe 16 biti
// returneaza adresa la partea low si high
#define ptrHIGH(X)      ( (byte*)(&(X))     )
#define ptrLOW(X)       (((byte*)(&(X))) + 1)

// definirea unei variabile pe 32biti care poate fi adresata ca 2 word sau 4
byte
```

```
typedef union {lword Lw; word w[2]; byte By[4];} blword;

#endif
```

Bibliografie

*** PCB83C552 Microcontroller User Manual, Philips, 1992.

*** C51 COMPILER User's Guide 11.93, Keil Elektronik GmbH, 1993.

*** A51 ASSEMBLER User's Guide 10.91, Keil Elektronik GmbH, 1993.

*** DScope 51 User's Guide 10.91, Keil Elektronik GmbH, 1991.

*** 8051 Utilities User's Guide 10.91, Keil Elektronik GmbH, 1991.

*** C167 16-Bit CMOS Single-Chip Microcontroller Data Sheet 06.94 Preliminary, Siemens, 1994.

*** C167 16-Bit Single-Chip Microcontroler User's Manual 08.94, Siemens, 1994.

*** C166 COMPILER User's Guide 4.92, Keil Elektronik GmbH, 1993.

*** A166 ASSEMBLER User's Guide 5.92, Keil Elektronik GmbH, 1992.

*** DScope 166 User's Guide 4.93, Keil Elektronik GmbH, 1993.

*** 80C166 Utilities User's Guide 4.93, Keil Elektronik GmbH, 1993.

*** The TTL Data Book for Design Engineers, Texas Instruments, 1976.

*** Design-in Reference Manual, Analog Devices inc., 1992.

*** Circuite integrate CMOS, Manual de utilizare, Microelectronica, Ed. Tehnică Bucureşti 1986.

*** MCB167, Prototype board with SIEMENS C167 CPU, User's Guide 3.94, Keil Software GmbH, 1994

*** Sistem de dezvoltare 80C552 Abacus SRL

*** CAN Specification, Version 2.0, Robert Bosch GmbH, 1991

*** CANPRES Version 2.0, Siemens Microelectronics, 1998

*** PCA82C250 CAN controller interface, Preliminary specification, Philips, 1997

*** P8xC592 8-bit microcontroller with on-chip CAN, Product specification, Philips, 1996

*** SJA1000 Stand-alone CAN controller, Preliminary specification, Philips, 1997

P.Buehring ş.a. Application of the P8xC592 microcontroller with CAN-interface, HKI/AN 91014, Hamburg, 1992.

H.C. Reuss Extended Frame Format - A New Option of the CAN Protocol (CAN Protocol Specification Vers. 2.0 A+B), HAI/AN 92 002, Hamburg, 1993

